easyMahout

Entorno de ejecución de algoritmos inteligentes de Mahout para Hadoop y Big Data

PROYECTO DE SISTEMAS INFORMÁTICOS
FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL
UNIVERSIDAD COMPLUTENSE DE MADRID

Director: Luis Garmendia Salvador
Codirector: José María Alcaraz Calero

Javier Sánchez González

Anghel Laurentiu Dulceanu

Daniel San Gabino Moreno

Contenido

Índice de figuras

LICENCIA

Resumen

"easyMahout" es un proyecto que tiene como objetivo hacer fácil lo difícil. Nos referimos a la utilización de algoritmos de minería de datos a través de Apache Mahout y Apache Hadoop. Hasta este momento, para utilizar las herramientas que nos ofrecían tanto Mahout como Hadoop necesitábamos de un conocimiento alto en lo referente al sistema operativo GNU/Linux, al uso de comando Shell y una gran inversión de tiempo en aprendizaje y configuración.

El objetivo de este proyecto es ofrecer al usuario una interfaz gráfica fácil, simple y sencilla, es decir, intuitiva. Aunque a primera vista podría parecer una interfaz demasiado simple, easyMahout esconde en sus entrañas una completísima funcionalidad y configuración de sus algoritmos, permitiéndonos hacer fácilmente, lo que hasta ahora era una serie interminable de comandos para construir sistemas de recomendación, agrupamiento o clasificación. El modo de empleo es directo, el usuario ofrecerá los datos de entrada a la aplicación, así como una serie de parámetros necesarios para la correcta ejecución de los mismos, y obtendrá los resultados. El usuario podrá cambiar ciertos parámetros desde la interfaz, ajustando los algoritmos a sus necesidades y podrá observar la variación de los resultados hasta encontrar la configuración óptima para sus datos.

La aplicación tiene la posibilidad de generar sistemas de recomendación, clustering y clasificación de datos genéricos de cualquier tipo, siempre y cuando tengan una estructura homogénea. El software “easyMahout” está dirigido a usuarios con cierto conocimiento en el uso de estos algoritmos de minería de datos, pudiendo así explotar el 100% de la funcionalidad que ofrece nuestra aplicación. Sin embargo, el funcionamiento es tan sencillo que cualquier persona será capaz de utilizarla con la ayuda de la lectura de esta memoria.

Una vez entendida la utilidad de la aplicación, también es importante explicar el ámbito en el que se encuentra. Hasta ahora, el lector podría pensar que no ofrecemos nada nuevo con respecto a otras aplicaciones de minería de datos. Esto es porque todavía no hemos introducido Apache Hadoop, ni el concepto de Big Data.

Apache Hadoop es un framework para trabajar con aplicaciones altamente distribuidas, es decir, trabajar con miles de nodos y petabytes de datos usando un relativamente nuevo paradigma de programación: MapReduce. ¿Qué ofrece nuestra aplicación que no ofrece ninguna otra? La posibilidad de ejecutar estos algoritmos escritos en MapReduce, con todos los beneficios que ello conlleva, a través de una aplicación fácil como la propuesta en este proyecto.

Palabras clave: Interfaz Gráfica, Recomendación, Clustering, Clasificación, Mahout, Hadoop, Minería, Big Data, MapReduce, Algorithmos Distribuidos

Abstract

"easyMahout" aims to make the use of distributed data mining algorithms available in Apache Mahout and Apache Hadoop easier. Currently, in order to use the algorithms offered by both Mahout and Hadoop, it is required a high knowledge of the GNU/Linux operating system, Shell command and a large investment in time discovering how to run and set up the framework.

The main aim of this project is to offer the user an intuitive, easy and simple graphical user interface. At first glance, it might seem too simple; easyMahout hides in her womb a very complete functionality and configuration of its algorithms, allowing easily for new functionalities which right now entails an endless series of commands to build recommender, clustering or classification systems. The usage of easyMohout is straightforward, the user provides the input data to the application as well as a number of parameters required for the proper execution of the same, and he gets the results.The user is able to change certain parameters adjusting the algorithms to better suits his needs and he will see the variation in the results to find the proper configuration for his data.

The application has the ability to generate recommender, clustering and classification systems of generic data of any kind, with the only restriction that they have to have a homogeneous structure. The "easyMahout" software is intended for users with some knowledge in the use of these data mining algorithms in order to exploit 100% of the functionality offered in easyMahout. However, the operation is so simple that anyone will be able to use it with the help of reading this document.

Once the utility of the application has been presented to the reader, it is also important to explain the context in which it is developed. The reader might think that this project does not offer something new to those other data mining applications. The reason is because we have not introduced Apache Hadoop, and Big Data concepts yet.

Apache Hadoop is a framework to work with highly distributed applications, like for instance, works with thousands of nodes and petabytes of data using a relatively new programming paradigm: MapReduce. What makes our application unique? The ability to run these algorithms written in MapReduce language, with all its benefits, through an easy application like ours.

Keywords: Graphic interface, Recommendation, Clustering, Classification, Mahout, Hadoop, Mining, Big Data, MapReduce, Distributed

1. Introducción

1.1. Algoritmos de Recomendación.

Cada día nos ocurren diversas cosas, escuchamos música, vemos series o películas en la televisión, probamos nuevos alimentos, etc. Inconscientemente, estamos creando una opinión personal sobre cada una de estas cosas. La canción X, del grupo Y que escuchamos por la radio nos pareció una maravilla. Probablemente esta nueva concepción no era una sorpresa, pues el grupo Y pertenece al mismo género musical que la mayoría de los grupos que escuchas. Además tenía muchas similitudes con el grupo Z, tu favorito. Las personas tendemos a apreciar cosas similares a las cosas que ya nos gustan. Por ésta razón, sería posible establecer criterios de elección, así como sistemas de recomendación que nos ayudaran en esta elección.

Otro claro ejemplo de un sistema de recomendación presente en nuestra vida cotidiana sería nuestro grupo de amigos amantes del cine, con los cuales compartimos la mayoría de gustos y opiniones sobre las películas que ambos hemos visto. De este modo, si un amigo ha ido recientemente al cine y le preguntas que película te recomienda para ir a ver, sin duda te recomendará la película que según vuestros gustos (la mayoría de ellos comunes) más susceptible sea de complacerte.

Como podemos observar, el proceso de formación de opiniones y de recomendación está muy presente en gran parte de la toma de decisiones a las que nos enfrentamos día a día. El sistema de recomendación al que más acostumbrados estamos a ver podría ser el de "Amazon", que nos sugiere productos para comprar, basados en las últimas búsquedas y las opiniones que nosotros mismos hemos dado sobre ciertos productos similares.

Como ya habremos deducido gracias a los ejemplos anteriores, un sistema de recomendación no es más que un sistema el cual recibe opiniones (preferencias) de distintos usuarios sobre distintos productos, y produce predicciones basadas en esos patrones.

Existen diferentes estrategias para descubrir nuevas cosas:

- Podríamos observar que objetos les gustan a otras personas que tienen gustos similares a nosotros. (Basados en usuario o *user-based recommenders*).
- También podríamos fijarnos en que objetos son similares a otros objetos que ya conocemos nosotros mismos. (Basados en objeto o i*tem-based recommenders*).

Los dos escenarios citados pertenecen, estrictamente hablando, al conjunto de los Filtros Colaborativos (Collaborative Filtering, CF) ya que producen recomendaciones basadas únicamente en el conocimiento de las relaciones de los usuarios con los objetos. La gran ventaja de este tipo de motores de recomendación es que no necesitamos ningún tipo de conocimiento acerca de las propiedades específicas de cada objeto.

La otra aproximación sería la que se basa en los atributos de los objetos, referida como técnicas de recomendación basadas en contenido (*content-based recommenders*). Un ejemplo sencillo es cuando un amigo te recomienda un grupo musical, porque es de Rock y cantado en español, tus principales preferencias. En esta ocasión se trata de una recomendación basada

en dos atributos de los grupos musicales: El género musical y el idioma. Este tipo de sistema de recomendación puede ser funcionar muy bien, pero es necesario un estudio exhaustivo del dominio del problema. Debido a esta razón, Mahout no se centra en éste tipo de sistema.

1.1.1. Construyendo la entrada del sistema de recomendación

Un sistema de recomendación recibe unos datos de entrada (de ahora en adelante, *input* o *preferencias*) y produce una salida, las recomendaciones. Una preferencia es una asociación de usuarios a objetos, y consiste en un identificador (*userID*) de usuario, en un identificador de objeto (*itemID*) y además un número que expresa el grado de preferencia del usuario a cierto objeto (*rating*). Este *rating* puede ser cualquier número real que exprese preferencias positivas, comúnmente valores de 1 (no le gusta) a 5 (le encanta).

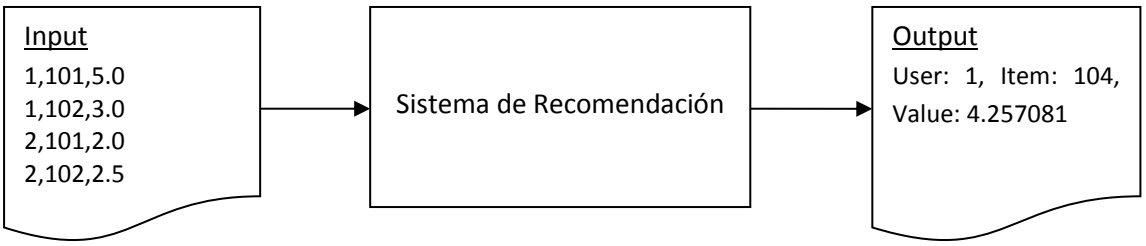


Figura 1: Flujo Recomendación.

El input del diagrama lo debemos interpretar como:

- El usuario 1 tiene una preferencia sobre el objeto 101 de 5.0
- El usuario 1 tiene una preferencia sobre el objeto 102 de 3.0
- El usuario 2 tiene una preferencia sobre el objeto 101 de 2.0
- El usuario 2 tiene una preferencia sobre el objeto 103 de 2.5

1.1.2. Sistema de recomendación basado en usuario a fondo

Un sistema de recomendación basado en usuario sigue un algoritmo bastante intuitivo:

```
Para cada ítem I para el cual U no tiene preferencia todavía
      Para el resto de usuarios V que si tienen preferencia para I
            Calcula la similitud S entre U y V
            Añadir las preferencias de V para I, ponderadas por S
Devolver los ítems con preferencias más altas.
```

El problema de este algoritmo es que sería extremadamente lento evaluar cada objeto, por ello primero calculamos un subconjunto con los usuarios más similares (*neighborhood*) y solo los objetos conocidos por este subconjunto de usuario son tenidos en cuenta.

```
Para cada usuario W distinto de U
      Calcula la similitud S entre U y V
      Guardar los usuarios con puntuación de similitudes más altas
Para cada ítem I que tiene preferencia de algún usuario en N
      pero U no tiene preferencia todavía
      Para el resto de usuarios V en N que tienen preferencia para I
            Calcula la similitud S entre U y V
            Añadir las preferencias de V para I, ponderadas por S
Devolver los ítems con preferencias de similitud más altas.
```

Este es el algoritmo que sigue un motor de recomendación basado en usuario estándar, y así esta implementado en Mahout.

1.1.3. Métricas de similitud de usuarios (UserSimilarity)

Una parte importante de los sistemas de recomendación es la elección de una métrica de similitud. Este componente soporta un gran peso del sistema. Existen muchas métricas de similitud y las más importantes están implementadas en Mahout.

Correlación de Pearson

La correlación de Pearson es un número entre -1 y 1 y mide la tendencia de dos series de números, emparejados uno a uno, de moverse juntos proporcionalmente. Es decir, es una relación lineal que mide el grado de relación de dos variables. Cuando esta tendencia es alta, la correlación está cerca de uno, sin embargo cuando la relación entre dos valores es pequeña, este valor se acerca a 0. Cuando esta relación tiende a ser negativa u opuesta, es decir, una serie de números es alta cuando otra serie es baja, el valor tiende hacia -1. Trasladado a nuestro contexto, la correlación de Pearson mide la tendencia de los valores de preferencias de dos usuarios a moverse juntos.

La correlación de Pearson no tiene en cuenta el número de objetos en los cuales dos usuarios coinciden, lo cual es una debilidad en el contexto de los motores de recomendación. Por ejemplo, dos usuarios con preferencias cada uno sobre 100 objetos (las cuales no coinciden entre usuarios) posiblemente tendrán mayor coeficiente de correlación que dos usuarios con solo 5 preferencias en común, aunque estas coincidan completamente. Para mitigar este

problema, podemos ponderar la correlación, es decir, darle un mayor peso positivo a aquellos valores de correlación que están compuestos por más ítems.

Otra característica presente es que no se puede calcular la correlación entre usuarios que solo tienen un ítem en común. Y por último, tampoco se puede obtener la correlación si todos los *ratings* de un usuario son iguales. (Suele ser un problema poco común).

Distancia Euclídea

Esta implementación de similitud se basa en la distancia entre usuarios, entendiendo a los usuarios como puntos en un espacio de muchas dimensiones (tantas dimensiones como objetos), cuyas coordenadas son los valores de preferencia. Esta métrica de similitud calcula la distancia Euclídea *d* entre dos puntos de usuarios. Pero este valor no es todavía una métrica valida, ya que los valores grandes significan más distancia, y por lo tanto menos similares. El resultado de la métrica debería ser más pequeño cuanto más similares son los usuarios.

Para ello, hacemos esta transformación:

Similitud = 1/(1+*d*) donde $\boldsymbol{d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}}$

Coseno

En esta nueva métrica también vemos los usuarios como puntos en un espacio n-dimensional. Ahora nos imaginemos dos líneas desde el origen (el punto (0,0,…,0)) hasta cada uno de estos dos puntos. Cuando dos usuarios son similares, tienen preferencias similares, por lo que estarán relativamente cerca en el espacio, o al menos tendrán una dirección semejante desde el origen. Por esta razón, el ángulo formado por estas dos líneas será pequeño si los usuarios son similares, o grande en caso contrario.

Utilizaremos el coseno de éste ángulo como medida de similitud. Si recordamos algo de trigonometría, sabremos que el coseno de un ángulo toma valores entre -1 y 1, y que el coseno de un ángulo pequeño estará cerca del 1, y el coseno de un ángulo cercano a 180 grados estará cerca del -1. Como podemos observar, el comportamiento del coseno encaja perfectamente en nuestra visión de métrica de similitud.

Correlación de Spearman

Para nuestros intereses, la correlación de Spearman es una variante de la correlación de Pearson. En vez de calcular la correlación basada en los valores de preferencia originales, usaremos la correlación basada en un ranking relativo de preferencias.

Calcular esta correlación para grandes conjuntos de datos puede ser muy costoso desde el punto de vista computacional, por lo que normalmente solo se suele utilizar con fines académicos y con pocos datos. Aun así, esta métrica puede dar resultados muy interesantes y valiosos.

Coeficiente de Tanimoto

Existen métricas de similitud que ignoran el valor de las preferencias de los usuarios, solo teniendo en cuenta si han expresado algún tipo de preferencia (buena o mala). El coeficiente de Tanimoto (también conocido como el coeficiente de Jaccard) hace uso de esta idea.

Esta medida es el número de objetos para el cual dos usuarios han expresado algún tipo de preferencia, dividido por el número de objetos que ambos usuarios han realizado preferencia.

Cuando los objetos de dos usuarios encajan perfectamente, el valor será 1, pero si los objetos no tienen nada en común el valor será 0. Como vemos, este valor nunca es negativo, pero no pasa nada, podemos extender el resultado a un rango de -1 a 1 mediante simples matemáticas:

Similitud = 2 * similitud -1

Solo se recomienda utilizar esta métrica si los valores de preferencia son booleanos o estos valores numéricos contienen mucho ruido e impiden un buen análisis de las preferencias.

Log-Likelihood

La última métrica que vamos a analizar es muy similar al coeficiente de Tanimoto., la cual tampoco tiene en cuenta los valores individuales de preferencia. La similitud resultante puede ser interpretada como la probabilidad de que dos objetos se solapen, pero sin esta coincidencia sea al azar.

Generalizando, podríamos decir que la similitud Log-Likelihood es una métrica más inteligente que el coeficiente de Tanimoto, aunque podría no ser así en todos los casos.

1.1.4. Neighboorhood

Como ya hemos comentado anteriormente, la necesidad del calcular subconjuntos de usuarios similares entre ellos viene obligada por ahorrar cálculos y mucho tiempo de ejecución. En Mahout tenemos dos maneras de crear estos subconjuntos: Nearest-N-Neighborhood y Threshold Neighborhood.

Nearest N Neighborhood

La idea no puede ser más sencilla, cogemos a los N usuarios más similares a cada usuario, creando un subconjunto para ellos. Lo único que puede darnos problemas es la correcta elección del número N de usuarios que formaran cada subconjunto. Un valor muy alto puede hacer que cojamos usuarios que no son lo suficiente similares, perdiendo eficacia en nuestras recomendaciones, pero el problema es similar si creamos subconjuntos de usuarios similares demasiado pequeños, pues podremos perder recomendaciones útiles.

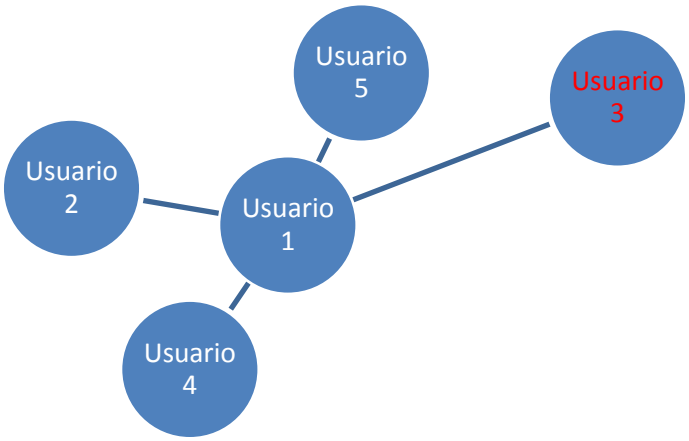


Figura 2: Neighborhood con N=3, donde el Usuario 3 se queda fuera.

La mejor manera de elegir este valor es realizar unas pequeñas pruebas sobre un conjunto de datos menor, y evaluar la calidad del sistema de recomendación, y así poder discernir el valor de N a usar.

Más adelante explicaremos como evaluar la calidad de los sistemas de recomendación, para poder realizar pequeñas pruebas con pocos datos para intentar configurar el sistema de la mejor manera posible para el tipo de datos en cuestión.

Threshold Neighborhood

Esta otra idea, también muy sencilla, se basa en decidir un umbral de similitud, a partir del cual decidiremos si un usuario es lo suficientemente similar a otro, como para formar parte de este subconjunto.

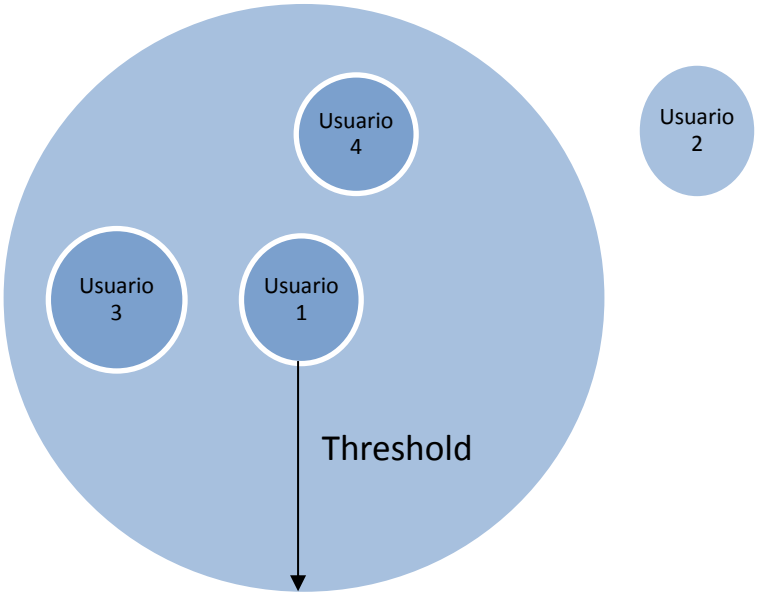


Figura 3: Neighborhood, donde el Usuario 2 se queda fuera por ser menos similar que el umbral.

El valor del umbral puede tomar valores entre 0 y 1, y de nuevo, encontrar el mejor valor será un trabajo de estudio para cada tipo de datos.

1.1.5. Sistema de recomendación basado en objeto.

Un sistema de recomendación basado en objeto funciona de manera muy similar a otro basado en usuario. Debemos configurar el modelo de datos y la similitud de ítems, que utiliza la misma implementación que la similitud entre usuarios. En este caso, no existe un ítem-neighborhood, ya que no necesitamos agrupar usuarios similares.

Dada esta gran similitud entre sistemas de recomendación, no necesitamos decir mucho más del funcionamiento de un sistema de recomendación basado en objetos.

1.1.6. Sistema de recomendación por factorización de matrices

La mayoría de los modelos de factorización están basados en el modelo factorial lineal. La matriz de valoraciones será modelada como el producto de la matriz de características de usuario y el matriz de características de los objetos.

$$R = U^T \times M$$

$$Dónde:$$

$$R = Matriz\ de\ valoraciones\ de\ usuario$$

$$U = Matriz\ de\ características\ de\ usuario$$

$$M = Matriz\ de\ características\ de\ los\ objetos$$

En Mahout encontramos dos tipos de factorización de matrices específicos para nuestras necesidades, estos son ALS-WR (Alternating Least Squares with Weighted-lambda-Regulations) y SVD (Descomposición de valores singulares, o Singular Value Decomposition en inglés).

ALS-WR

Este método trata de solventar el principal problema de los algoritmos SVD estándar, donde cuando faltan muchos elementos en la matriz R, no pueden encontrar satisfactoriamente las matrices U y M.

Algoritmo ALS:

1. Inicializar la matriz M asignando la media de las valoraciones para cada objeto en la primera fila. Completar la matriz con valores aleatorios pequeños.
2. Actualiza M, soluciona U minimizando la función objetivo (la suma de los errores al cuadrado).
3. Actualiza U, soluciona M minimizando la función de similitud objetivo.
4. Repite los pasos 2 y 3 hasta que el criterio de parada sea satisfecho.

Este criterio de parada se basa en las observaciones de RMSE (Root Mean Square Error) sobre un subconjunto de prueba. Después de una iteración, si la diferencia entre los RMSE observados en el conjunto de prueba es menor que 0.0001, paramos la ejecución de algoritmo y utilizaremos la matrices U y M obtenidas en esta iteración para hacer las predicciones finales sobre el conjunto de test.

Para evitar el overfitting (o sobreajuste), el algoritmo utiliza una regulación λ ponderada. El valor óptimo de λ será determinado mediante prueba y error. De la publicación "Large-scale Parallel Collaborative Filtering for the Netflix Prize" sacamos el valor óptimo $\lambda = 0.065$.

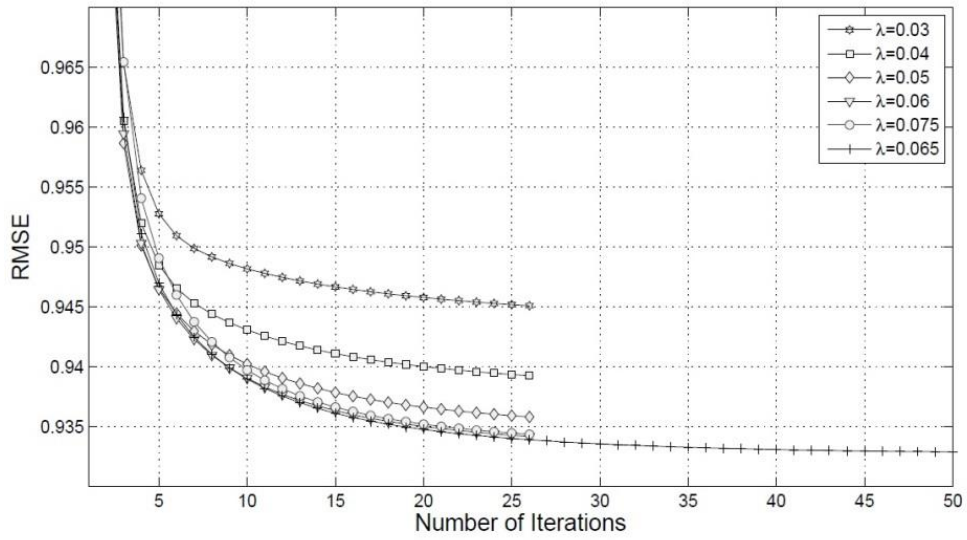


Figura 4: Variación de RMSE según λ y el número de iteraciones del algoritmo. Ref. [13]

1.1.7. Evaluando un sistema de recomendación

Un motor de recomendación en una mera herramienta que nos permite contestar a la pregunta: ¿Cuáles son las mejores recomendaciones para un usuario? Pero, ¿Qué es una buena recomendación? ¿Cómo sabemos si estamos produciendo una buena recomendación?

El mejor sistema posible sabría de algún modo, antes de que el usuario asignara un valor de preferencia, cuál sería el valor exacto, por lo tanto, podría predecir cualquier preferencia futura. De hecho, los sistemas de recomendación funcionan así, estimando preferencias para los objetos que el usuario no las ha proporcionado. Una manera de evaluar estos sistemas es evaluar la calidad de esas estimaciones midiendo como de cerca están las estimaciones respecto a los valores reales.

Como es obvio, un sistema de recomendación no dispone de todos los valores de preferencias reales (ya que su trabajo es estimar estos valores), por lo que tomaremos una pequeña porción de los datos de entrada como datos de test. Estos datos de test los quitamos de los datos de entrada, e intentaremos estimas los valores de preferencias que hemos separado, para después compararlos los estimados con los reales.

Ahora es simple producir una puntuación de calidad del sistema de recomendación. Podemos tomar la media de la diferencia entre el valor estimado y el real, donde cuanto más pequeño es este valor, mejores son las estimaciones. Una puntuación de 0.0 significaría una estimación perfecta, no existe diferencia entre todos los valores estimados y los reales.

En algunos casos es usada la media cuadrática (RMS), que no es otra cosa que la raíz cuadrada de la media de los cuadrados de las diferencias entre las estimaciones y los datos reales. Una vez más, valores pequeños significan mejores.

Mahout nos brinda la implementación del evaluador, solo tenemos que decidir qué tipo de medida de error usar, y el tamaño en porcentaje de los datos de prueba.

Existen otras formas de evaluar la calidad de un sistema de recomendación. Por ejemplo, podemos utilizar métricas clásicas en la recuperación de información, como los motores de búsqueda. Estas son la precisión y la exhaustividad (precision and recall, en inglés).

La precisión es la fracción de instancias recuperadas que son relevantes, mientras la exhaustividad es la fracción de instancias relevantes que han sido recuperadas. Un ejemplo en el que participa un motor de búsqueda que, ante una consulta dada, retorna 30 páginas de las cuales sólo 20 son relevantes dejando 40 páginas relevantes fuera de la búsqueda. Este motor tendrá entonces una precisión de 20/30 = 2/3 mientras que su exhaustividad es 20/60 = 1/3.

Estos parámetros pueden ser trasladados fácilmente a nuestros intereses: la precisión es la proporción de las recomendaciones generadas que son buenas, mientras que el recall es la proporción de buenas recomendaciones que han sido devueltas.

Por ejemplo, si la precisión calculada es 0.75, significa el ¾ de las recomendaciones hechas fueron buenas. Si la exhaustividad es 1.0, todas las buenas recomendaciones posibles fueron recomendadas.

1.1.8. Tabla resumen

Sistema de recomendación	Parámetros clave	Características clave
Basado en usuario	• Métrica de similitud de usuarios • Definición y tamaño del Neighborhood	• Rápido cuando el número de usuarios es relativamente pequeño
Basado en objeto	• Métrica de similitud de objetos	• Rápido cuando el número de ítems es relativamente pequeño
Factorización de matrices	• Número de características	• Buenos resultados • Requiere una costosa computación de las matrices

Figura 5: Tabla tipos de sistemas de recomendación.

1.2. Algoritmos de clústeres

1.2.1. Introducción al clustering

Como seres humanos, tendemos a asociar personas que piensan igual. Tenemos una gran capacidad mental para encontrar patrones que se repiten, y continuamente asociamos lo que vemos, oímos, olemos, saboreamos con las cosas que ya están en nuestra memoria. Por ejemplo, el sabor de la miel nos recuerda más al sabor del azúcar que de la sal. Así que agrupamos las cosas que tienen un sabor parecido al azúcar y miel y los llamamos dulces. Sin saber cómo sabe el dulce, sabemos que todas las cosas dulces en el mundo son similares y pertenecen al mismo grupo. También sabemos lo diferentes que son de todas las cosas que pertenecen al grupo salado. Inconscientemente, agrupamos en conjuntos los sabores en estos *clústeres*. Tenemos clústeres con cosas dulces y clústeres con cosas saladas, cada grupo teniendo cientos de artículos en él.

En la naturaleza, observamos muchos otros tipos de grupos. Considere los simios y los monos, que son ambos tipos de primates. Todos los monos comparten algunos rasgos como una altura más corta, una larga cola, y una nariz plana. Por otro lado, los monos tienen un tamaño más grande, brazos largos y cabezas grandes. Los simios son diferentes de los monos, pero a ambos les gustan los plátanos. Así que podemos pensar en los simios y los monos como dos grupos diferentes o como un solo grupo de primates amante de plátano. Lo que consideramos como un clúster depende totalmente de los rasgos que elijamos para medir la similitud entre objetos (en este caso, los primates). En este capítulo, vamos a explicar cómo funciona el proceso de clustering utilizando los algoritmos de Mahout por un lado y Hadoop (datos en ficheros distribuidos en distintos DataNodes) por otro.

Entonces, ¿cuál es el proceso de agrupar todo esto? Suponga que le dan las llaves de un biblioteca con miles de libros. Los libros fueron colocados en ningún orden predeterminado. Los lectores que entran en la biblioteca tendrían que mirar a través de todos los libros, uno por uno, para encontrar un libro en particular. Esto no sólo es engorroso y lento, es aburrido también. Una clasificación de los libros en orden alfabético por título sería una gran mejora para los lectores.

¿Qué pasa si la mayoría de la gente estaba simplemente navegando o investigando sobre un tema general? La agrupación de los libros por tema sería más útil que un orden alfabético por título. Pero, ¿cómo se puede incluso comenzar esta agrupación? Habiendo tomado este trabajo, ni siquiera puede estar seguro de cuáles eran todos los libros sobre un tema en particular. Para agrupar los libros por tema, podrían poner todos en una línea y comenzar a leer uno por uno. Cuando se encuentra con un libro cuyo contenido es similar a un anterior libro, puede volver atrás y apilarlos juntos.

Cuando haya terminado, tendrá cientos de pilas de libros en lugar de miles de libros individuales. Si un centenar de grupos temáticos está de más, podría ir de nuevo al principio de la línea y repetir el proceso con las pilas, hasta que sus pilas serán sobre temas muy diferentes unos de otros. El clustering tiene que ver con la organización de los elementos de una colección dada en grupos de artículos similares. Estos grupos podrían ser considerados

como conjuntos de elementos similares entre sí en algunos aspectos, pero diferentes de los elementos que pertenecen a otros grupos.

La agrupación de una colección implica tres cosas:

- Un algoritmo: Este es el método utilizado para agrupar los libros juntos.
- Una noción tanto de similitud como de disimilitud: En la discusión anterior, nos basamos en su evaluación de los libros que pertenecían a una pila existente, y que debe comenzar una nueva.
- Una condición de parada: En la biblioteca del ejemplo, este podría ser el punto más allá del que los libros no se pueden apilar más, o cuando las pilas ya están bastante disímiles.

Hasta ahora, hemos pensado en los elementos de clústeres como apilarlas, pero en realidad era simplemente agruparlas. Conceptualmente, la agrupación es más como mirar a los grupos que forman los elementos parecidos entre sí y rodeándolos. La figura 6 muestra la agrupación de puntos en un plano estándar X-Y. Cada círculo representa un clúster que contiene varios puntos.

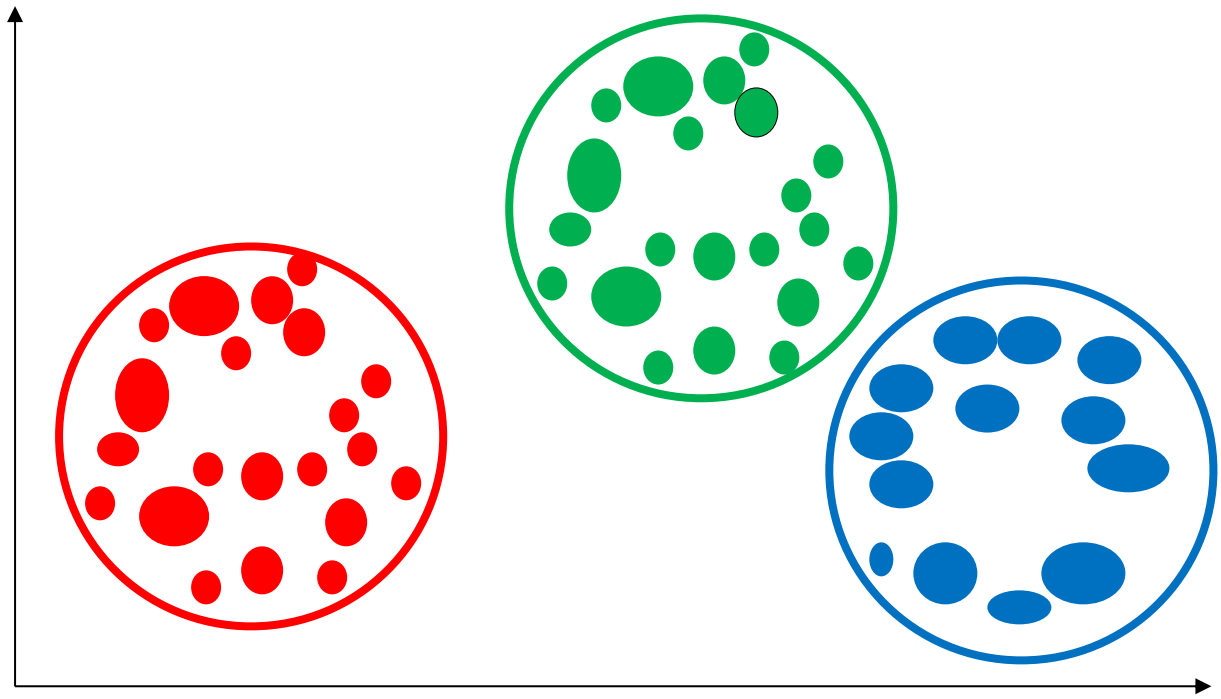


Figura 6: Clústeres en el plano.

En este ejemplo sencillo, los círculos representan obviamente la mejor agrupación de puntos en tres grupos basados en la distancia. Los círculos son una buena manera de pensar en los clústeres, ya que las agrupaciones también se definen por un punto central y el radio. El centro del círculo se llama el centro de gravedad, o media (promedio) , de esa agrupación. Es el punto cuyas coordenadas son el promedio de las coordenadas x e y de todos los puntos en el clúster. En este capítulo, visualizamos la agrupación como un problema geométrico. También

utilizamos técnicas de la geometría para explicar las diversas medidas de distancia, que son fundamentales para algoritmos de clustering.

También tenemos en cuenta algunas de las importantes medidas de distancia y sus relaciones con el clustering. Esto le ayudará a visualizar la similitud aguda entre los puntos de agrupamiento en un plano y el texto de agrupación de los documentos.

Después veremos algunos de los métodos que se utilizan popularmente para el clustering de datos y la forma en que se implementan en Mahout. La estrategia en los ejemplos de la biblioteca era fusionar las pilas de libros hasta que se llegó a un umbral. El número de grupos formados en este ejemplo dependía de los datos; basado en el número de libros y el umbral, que podría haber terminado con 100, 20, o incluso sólo 1 clúster. Una estrategia más popular es la de establecer un número de grupos, en lugar de un umbral y, a continuación, busque la mejor agrupación con esa restricción.

1.2.2. La medición de la similitud de datos

La cuestión más importante en la agrupación es encontrar una función que cuantifica la similitud entre dos puntos de datos como un número. En el ejemplo de plano *xy*, la medida de similitud (la similitud métrica) para los puntos era la distancia euclidiana entre dos puntos. En el ejemplo de la biblioteca no teníamos tal medida clara, matemática y en su lugar nos basábamos enteramente en la sabiduría de la bibliotecaria para juzgar la similitud entre los libros. Una posible métrica podría basarse en el número de palabras comunes a dos títulos de libros. Por ejemplo, *Harry Potter: La Piedra Filosofal* y *Harry Potter: Prisionero de Azkaban* tiene dos palabras en común: *Harry* y *Potter*. A pesar de que *El Señor de los Anillos: Las Dos Torres* es similar a la serie de Harry Potter, esta medida de la similitud no lo capta en absoluto. Se necesitaría alterar la métrica de similitud para tener en cuenta el contenido de los libros. Se podría contar las palabras que aparecen en cada libro y cuando se tendrá números de palabras cercanos los unos a los otros se podrá decidir que los libros son similares. Desafortunadamente, no es tan fácil hacerlo. No sólo estos libros tienen cientos de páginas de texto, pero las características del idioma también influye ya que dependiendo de las palabras que más veces se repiten en un idioma sabremos que esas palabras no serán relevantes para llevar a cabo el proceso de clustering de los libros. Habría que usar una medida de similitud de los libros en la que tener en cuenta una media ponderada de cada palabra del libro.

Estos y muchos otros trucos son parte de un método de ponderación popular llamado TF –IDF (término de frecuencia inversa documento frecuencia). A continuación vamos a explicar todos los pasos necesarios para ejecutar una tarea de clustering en nuestra interfaz gráfica.

Antes de empezar es muy importante saber qué es lo que queremos hacer con nuestros datos, es decir, qué algoritmo vamos a usar, qué medida de similitud, qué umbral de convergencia, así como también, dependiendo de la situación, número de clústeres que necesita que la aplicación le devuelva y el número de iteraciones que el algoritmo empleará. Primero nos centraremos en los algoritmos de Mahout, para después explicar cómo ejecutar una tarea de clustering usando Hadoop.

Asi, en primer lugar habrá que elegir el algoritmo (Algorithm) que queremos usar para la tarea. Las posibilidades son:

- K-Means
- Canopy
- Fuzzy K-Means

K-Means

Es un algoritmo de clustering genérico que puede ser moldeado fácilmente a adaptarse a casi todas las situaciones. También es fácil de entender y puede ser fácilmente ejecutado en el ordenador. Antes de entrar en los detalles de los diferentes algoritmos de clustering, lo mejor es conseguir un poco de experiencia práctica en el uso del algoritmo K-Means. Entonces entenderemos mejor las deficiencias y las dificultades que nos ofrecen otras técnicas menos genéricas.

K- Means es al clustering lo que el jarabe es para la tos. Es un algoritmo simple y tiene una antigüedad de más de 50 años. Stuart Lloyd propuso por primera vez el algoritmo estándar en 1957 como técnica de modulación por impulsos codificados, pero no fue hasta 1982 cuando lo intentó publicar. Es ampliamente utilizado como un algoritmo de clustering en muchos campos de la ciencia.

El algoritmo requiere que el usuario establezca el número de grupos (clústeres), K, como parámetro de entrada. Echemos un vistazo a los detalles para entender el algoritmo. El algoritmo K-Means pone una limitación de hardware sobre el número de clústeres, K. Esta limitación podría hacer que dude de la calidad de este método, pero este algoritmo ha demostrado que funciona muy bien para una amplia gama de problemas del mundo real durante los 33 años de su existencia. Incluso si su estimación para el valor de K no es óptima, la calidad del clustering no se ve afectada tanto por ella. Supongamos que estamos agrupando artículos de noticias para obtener categorías de nivel superior como la política, la ciencia y los deportes. Para tal efecto es probable que elijamos un valor pequeño de K, tal vez en el rango de 10 a 20, todo ello dependiendo de lo mucho que queremos afinar.

Supongamos que hay 1.000.000 artículos de noticias en la base de datos y que estamos tratando de encontrar grupos de artículos sobre la misma historia. El número de tales historias relacionadas sería mucho menor que el corpus entero, tal vez en el rango de 100 artículos por clúster.

Esto significa que se necesita un valor K de 10 000 para generar una distribución de este tipo. Eso seguramente pondrá a prueba la capacidad de ampliación del clustering, y aquí es donde brilla Mahout. Para un clustering de buena calidad utilizando K-Means, tendremos que calcular un valor para K. Una forma aproximada de estimar K es basándonos en los datos que tenemos y el tamaño de los clústeres que necesita. En el ejemplo anterior, donde tenemos alrededor de un millón de artículos de prensa, si hay un promedio de 500 noticias publicadas sobre cada historia única, debemos comenzar el clustering con un valor K de aproximadamente 2000 (1000000 /500).

Veamos el algoritmo K-Means en detalle. Supongamos que tenemos n puntos que necesitamos a agruparse en K grupos. El algoritmo K-Means se iniciará con un conjunto inicial de K centroides. El algoritmo hace múltiples rondas de procesamiento y refina el centroide hasta que se alcance el max-iteraciones o hasta que los centroides convergen en un punto fijo del que no se mueven mucho.

Una única iteración K-Means se ilustra en la figura 2. El algoritmo real es una serie de tales iteraciones.

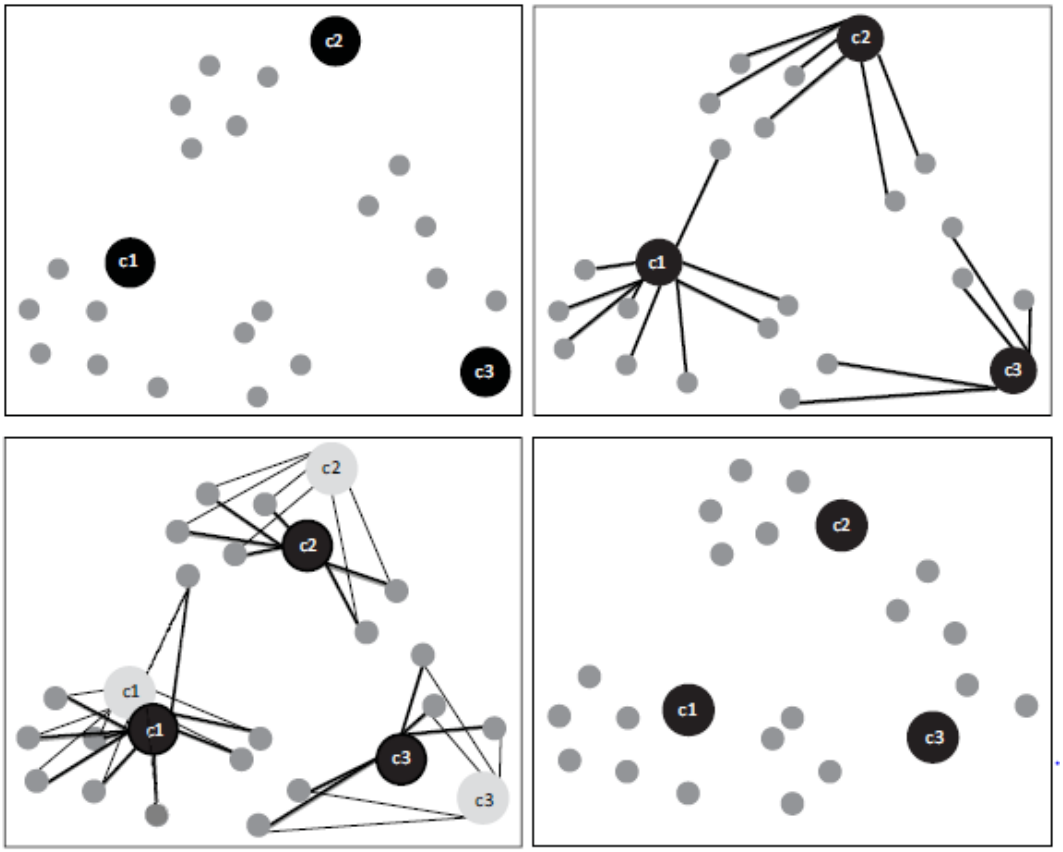


Figura 7: K-Means clustering en acción. Ref. [27]

A partir de tres puntos aleatorios como centroides (arriba a la izquierda), la etapa map (parte superior derecha) asigna a cada punto al clúster más cercano a él. En la etapa reduce (izquierda inferior), los puntos asociados se promedian para producir la nueva ubicación del centro de gravedad, dejándoles con la configuración final (parte inferior derecha). Después de cada iteración, la configuración final se alimenta de nuevo en el mismo bucle hasta que los centroides se encuentran en sus posiciones finales.

Canopy

Es un algoritmo que se distingue de K-Means en el hecho de que no necesita el parámetro número de clústeres definido por el usuario sino que es el propio algoritmo el que calcula el número adecuado de clústeres.

La generación del Canopy, también conocido como clustering Canopy, es una técnica rápida de clustering aproximado. Se utiliza para dividir el conjunto de entrada de puntos en grupos superpuestos conocidos como canopies. La palabra canopy, en este contexto, se refiere a un círculo de puntos, o un cluster. El algoritmo de clustering Canopy intenta estimar los centroides de grupo aproximadas (o canopy-centroides) utilizando dos umbrales de distancia.

La fuerza del clustering por medio del algoritmo Canopy reside en su capacidad para crear grupos extremadamente rápido, se puede hacer esto con un solo pase de los datos. Pero su fuerza es también su debilidad. Este algoritmo no puede dar clústeres exactos y precisos. Pero se puede dar el número óptimo de las agrupaciones sin siquiera especificar el número de clústeres, K, como se requiere por K-Means.

El algoritmo utiliza una medida de distancia rápida y dos umbrales de distancia, T1 y T2, con T1 > T2. Se inicia con un conjunto de datos de puntos y una lista vacía de canopies, y luego itera sobre el conjunto de datos, la creación de canopies en el proceso. Durante cada iteración, se elimina un punto a partir del conjunto de datos y agrega un canopy a la lista con ese punto como el centro. Se recorre el resto de los puntos de uno en uno. Para cada uno, se calcula las distancias a todos los centros de canopy en la lista y si la distancia entre el punto y cualquier centro de canopy está dentro de T1, se añade en ese canopy. Si la distancia está dentro de T2, es removido de la lista y por lo tanto impide la formación de un nuevo canopy en los bucles posteriores. Repite este proceso hasta que la lista está vacía. Este enfoque evita que todos los puntos cercanos a un canopy ya existente (distancia < T2) sean el centro de un nuevo canopy. Es perjudicial formar otro canopy redundante en las proximidades de uno ya existente. La Figura 8 ilustra los canopy creados usando este método. Qué grupos se forman sólo depende de la elección de los umbrales de distancia.

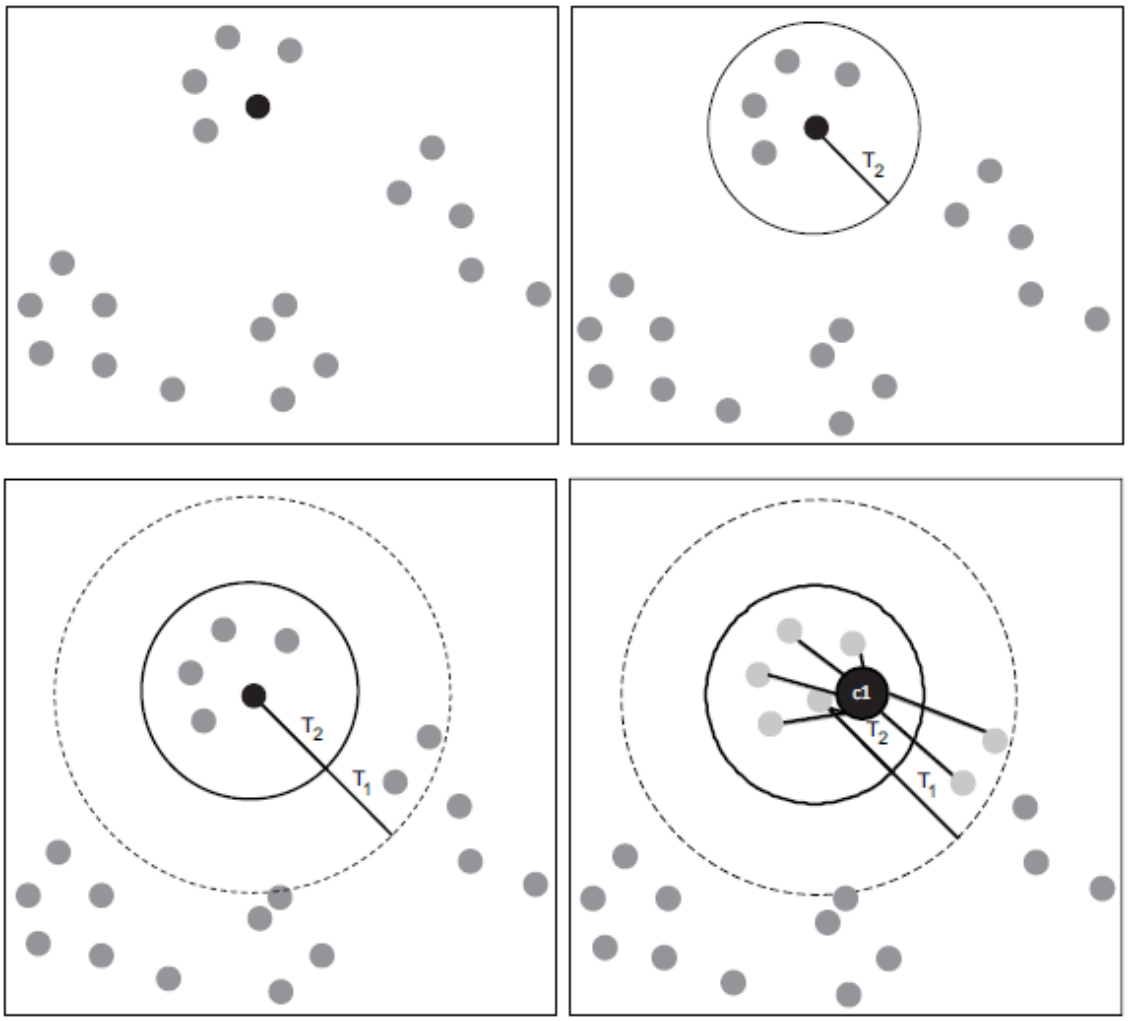


Figura 8: Canopy-clustering en acción. Ref. [27]

Clustering Canopy: Si comienza con un punto (arriba a la izquierda) todos los puntos dentro de la distancia T2 (arriba a la derecha) se eliminan del conjunto de datos y se les impide convertirse en nuevos canopies. Los puntos dentro del círculo exterior (parte inferior izquierda) también se ponen en el mismo canopy, pero se les permite ser parte de otros canopies. Este proceso de asignación se realiza en un mapper de un solo paso. El reductor calcula el promedio del centroide (abajo a la derecha) y se elimina canopies cercanos.

Fuzzy K-Means

Como su nombre lo indica, el algoritmo K-Means de clustering difuso invoca una forma difusa de clustering K-Means. En lugar de la agrupación exclusiva de K-Means, el "K-Means borroso" intenta generar clústeres solapados del conjunto de datos. En la comunidad académica, también es conocido como el algoritmo Fuzzy K-Means. Puede pensar en él como una extensión de K-Means. K-Means intenta encontrar los clústeres duros (donde cada punto pertenece a un clúster) mientras que Fuzzy K-Means descubre los grupos blandos. En un clúster suave, cualquier punto puede pertenecer a más de un grupo con un cierto valor de afinidad hacia cada uno. Esta afinidad es proporcional a la distancia desde el punto al

centroide del clúster. Como K-Means, Fuzzy K-Means trabaja en esos objetos que se pueden representar en espacio vectorial de n dimensiones y tiene una medida de distancia definida.

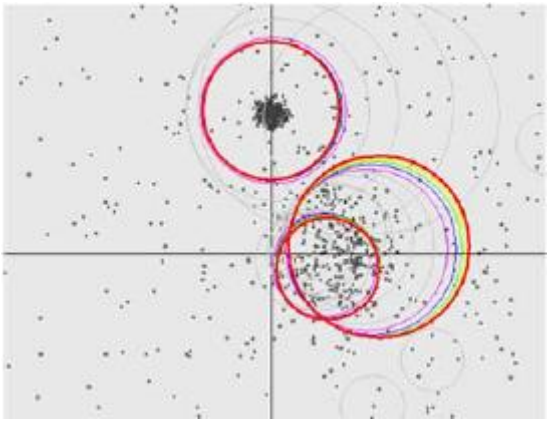


Figura 9: Muestra visual ejecución Fuzzy K-Means.

Fuzzy K-Means clustering: Los grupos parece que se superponen entre sí, y el grado de solapamiento se decide por el parámetro de borrosidad.

¿Cómo de difusa es demasiado difusa?

K-Means Fuzzy tiene un parámetro, m, llamado el factor de borrosidad (fuzzyfication factor). Al igual que K-Means, Fuzzy K-Means itera los bucles sobre el conjunto de datos, pero en lugar de asignar los vectores a los centroides más cercanos, se calcula el grado de asociación del punto a cada uno de los grupos.

Supongamos que tenemos un vector, V y que d1, d2,... dk son las distancias a cada uno de los clústeres de K centroides. Se calcula el grado de asociación (U1) del vector (V) al primer clúster (C1) como

$$u_i = \frac{1}{\left(\frac{d_1}{d_1}\right)^{\frac{2}{m-1}} + \left(\frac{d_1}{d_2}\right)^{\frac{2}{m-1}} + \cdots + \left(\frac{d_1}{d_k}\right)^{\frac{2}{m-1}}}$$

Del mismo modo, se puede calcular el grado de asociación con otros grupos mediante la sustitución de d1 en los numeradores de la expresión anterior con D2, D3, y así sucesivamente. Está claro a partir de la expresión que m debe ser mayor que 1, o bien el denominador de la fracción se convierte en 0 y las cosas se rompen.
Si elige un valor de 2 para m, verá que todos los grados de asociación para cualquier punto se pueden resumir a uno. Si, por otro lado, m viene siendo muy cerca de 1, como 1.000001, se le dará más importancia al centroide más cercano al vector. El algoritmo Fuzzy K-Means comienza a comportarse más como el algoritmo K-Means a la vez que m se acerca a 1.

Si *m* aumenta, la falta de claridad del algoritmo aumenta, y comenzará a ver más y más solapamiento. El algoritmo Fuzzy K-Means también converge mejor y más rápido que el algoritmo K-Means estándar.

1.2.3. Función de similitud (Distance Measure)

La otra variable importante es la función de similitud, que es la manera de encontrar la proximidad de un punto a los centroides de grupo. Nos gustaría señalar que Mahout también proporciona otra manera de calcular las distancias:

- La distancia entre dos vectores de doble valor, respectivamente, X_i, Y_i se denomina como *SquaredEuclideanDistanceMeasure*; La fórmula para esto es la siguiente:

$$d(x,y) = (x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2$$

- La distancia conocida como *ManhattanDistanceMeasure* se calcula siguiendo la fórmula:

$$d(x,y) = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n|$$

- La distancia conocida como *CosineDistanceMeasure* se calcula siguiendo la fórmula:

$$d(x,y) = \frac{\sum_{i=1}^{n} x_i * y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} * \sqrt{\sum_{i=1}^{n} y_i^2}}$$

- La *TanimotoDistanceMeasure* sigue la siguiente fórmula:

$$d(x,y) = \frac{\sum_{i=1}^{n} x_i * y_i}{\sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} y_i^2 - \sum_{i=1}^{n} x_i * y_i}$$

- La distancia *WeightedDistanceMeasure* se mide con la fórmula:

$$d(x,y) = \sqrt{\sum_{i=1}^{n} \left(\frac{x_i}{s_i} - \frac{y_i}{s_i}\right)^2}$$

Umbral de convergencia (Convergence Threshold).
Es un valor numérico entre 0 y 1, que se usa como parámetro de entrada para el algoritmo invocado en cada caso.

1.2.4. Número de clusters (Number of Clusters)

Es un parámetro entero positivo cuyo valor se usa especialmente para ejecutar el algoritmo K-Means y obtener los k primeros centros de clústeres que el algoritmo usará en las próximas iteraciones. Este parámetro no está disponible para el algoritmo Canopy.

1.2.5. Máximo número de iteraciones (Maximum Iterations)

Es un número entero positivo que impone un límite en las repeticiones de ejecución de los algoritmos.

1.2.6. Modelo de datos (DataModel)

Vamos a usar solo el tipo de fichero Extended que es el único que permite delimitar los datos entre sí. Lo más importante acerca de los datos de entrada, es el saber que antes de cada ejecución de un algoritmo dichos datos están siendo preprocesados y procesados para que se pueda invocar a un algoritmo Mahout. A continuación vamos a explicar en detalle el proceso de tratamiento de los datos.

Para obtener un buen agrupamiento, es necesario comprender las técnicas de vectorización: el proceso de representar los objetos como vectores. Un vector es una representación muy simplificada de datos que puede ayudar a entender los algoritmos de clustering y a calcular su similitud con otros objetos. A continuación analizaremos varias formas de convertir diferentes tipos de objetos en vectores. Los libros se agrupan sobre la base de la similitud de sus palabras, y los puntos en un plano de dos dimensiones estaban agrupados juntos en base a las distancias entre ellos. En realidad, la agrupación se podría aplicar a cualquier tipo de objeto, con la única restricción de que se pueda distinguir artículos similares y distintos. Las imágenes pueden ser agrupadas en función de sus colores, las formas de las imágenes, o por las personas que salen en ellas. Podríamos agrupar fotografías para quizás tratar de distinguir las fotos de los animales de los de los humanos. Podríamos incluso agrupar especies de los animales por su tamaño promedio, pesos y número de patas para descubrir agrupaciones automáticamente.

1.2.7. Visualización de vectores

Es posible que haya encontrado la palabra vector en muchos contextos. En física, un vector denota la dirección y magnitud de una fuerza, o la velocidad de un objeto en movimiento como un coche. En matemáticas, un vector es simplemente un punto en el espacio. Ambas representaciones son conceptualmente muy similares. En dos dimensiones, los vectores se representan como una lista ordenada de valores, uno para cada dimensión, como (4, 3). Ambas representaciones se ilustran en la figura 5. Nosotros a menudo ponemos el nombre de la primera dimensión *X* y el segundo *Y* cuando se trata de dos dimensiones, pero esto no importa para nuestro propósito en Mahout. En lo que a nosotros respecta, un vector puede tener 2, 3, o 10 000 dimensiones. La primera dimensión es 0, la siguiente es la dimensión 1, y así sucesivamente.

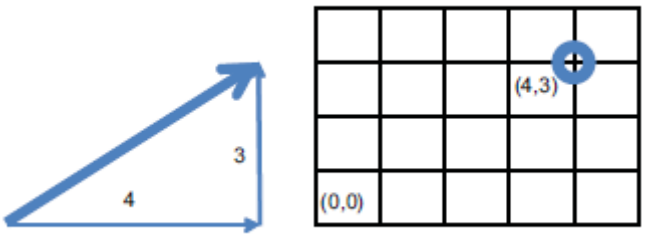


Figura 10: Representación vector 2D físicos y matemáticos. Ref. [27]

1.2.8. La transformación de los datos en vectores

En Mahout, los vectores se implementan como tres clases diferentes, cada uno de los cuales está optimizado para diferentes escenarios: *DenseVector, RandomAccessSparseVector y SequentialAccessSparseVector*.

- *DenseVector* puede ser pensado como una serie de números reales, cuyo tamaño es el número de características de los datos. Debido a que todas las entradas de la matriz se preasignan independientemente de si el valor es 0 o no, se llama denso.

- *RandomAccessSparseVector* se implementa como un HashMap entre un número entero y un real, donde se asignan los únicos rasgos que no sean cero. Por lo tanto, son llamados como SparseVectors.

- *SequentialAccessSparseVector* se implementa como dos conjuntos paralelos, uno de los enteros y el otro de reales. Sólo las entradas distintas de cero se guardan en él. A diferencia del RandomAccessSparseVector, que está optimizado para el acceso aleatorio, éste está optimizado para la lectura lineal.

1.2.9. Preparación de vectores para su uso por Mahout

Echemos un vistazo a la forma de preparar los vectores específicamente para el consumo por los algoritmos Mahout. Una implementación de Vector es una instancia rellenado para cada objeto. Todos los vectores se escriben en un archivo en el formato *SequenceFile*, que es leído por los algoritmos de Mahout.

SequenceFile es un formato de la biblioteca de Hadoop que codifica una serie de pares clave-valor. Las claves deben implementar *WritableComparable* de Hadoop, para poder compararlas entre sí, y los valores deben implementar Writable. Estos son los equivalentes Hadoop a las interfaces Comparable y *Serializable* de Java. Por ejemplo, vamos a usar el nombre o la descripción del vector como una clave, y el vector a sí mismo como el valor. Las clases vectoriales de Mahout no implementan la interfaz de escritura para evitar acoplar directamente a Hadoop, pero la clase contenedora *VectorWritable* puede ser utilizada para envolver un Vector y permitir su escritura. El Mahout vectorial puede escribir el fichero *SequenceFile* utilizando la clase *VectorWritable*. La clase utilizada para convertir cualquier fichero de datos a vectores es, en nuestra aplicación *CreateSequenceFile*.

1.3. Algoritmos de clasificación

Imagina la siguiente situación, aunque es obvio para nosotros a simple vista, se trata de un ejemplo para entender de qué se trata un clasificador: Tenemos varias manzanas de las cuales tres son verdes y cinco rojas, también tenemos cuatro peras que son verdes. Si cogiéramos una al azar sin verla, a partir de solo datos no podríamos saber si una fruta que es verde sería una pera o una manzana pero si fuera roja sabríamos que es una manzana. La solución sería introducir más variables para poder saberlo, por ejemplo la forma, sabiendo si es redonda o alargada podríamos decir si es una pera o una manzana. Con la forma solo tampoco podríamos averiguar si se trata de un tipo de manzana u otra, necesitaríamos más datos.

La clasificación se trata de esto, a partir de ciertas características sobre un elemento intentar averiguar otra. Dependiendo del atributo a identificar necesitaremos unas características u otras, en otros casos algunas no nos servirán para calcularlo como podría ser el peso en el ejemplo anterior ya que tienen similares pesos y no es un característica distintiva entre peras y manzanas.

Los algoritmos de clasificación son algoritmos de aprendizaje supervisado, en los que a partir de ejemplos se calculan los patrones que siguen para obtener un modelo. Este modelo sirve para calcular alguna característica que tengan estos ejemplos sobre una entrada de la que no se sabe el valor de esa característica. El proceso sería pasarle la entrada en el formato requerido (dependiendo del algoritmo) al algoritmo y este se encarga de producir dicho modelo, una vez creado este podemos introducir nuevas entradas al modelo con la característica desconocida a calcular. Aquí podemos ver un ejemplo:

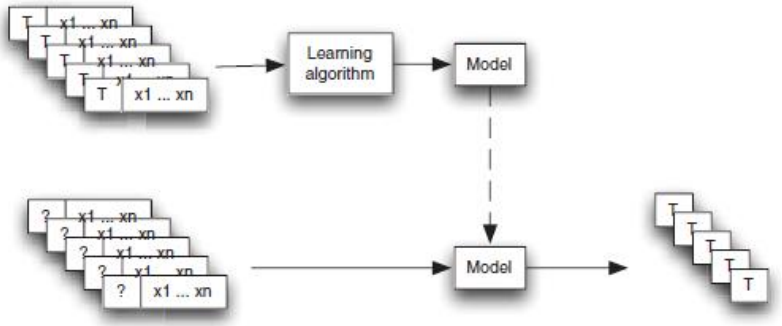


Figura 11 : Clasificador, aprendizaje

Mahout puede usarse en muchos proyectos de clasificación, pero la ventaja de usarlo reside en que el número de los ejemplos de entrenamiento es demasiado grande, varía bastante. Hasta aproximadamente 100.000 ejemplos otros sistemas de clasificación son eficientes, pero como generalmente la entrada se encuentra entre 1 millón y 10 millones algo escalable como Mahout es requerido.

Mahout es adecuado para cuando los datos son más grandes o más rápido crecen y donde tras soluciones son menos factibles:

Tamaño del sistema (nº de ejemplos)	Elección del método de clasificación
< 100K	Tradicional. Los métodos fuera de Mahout deberían funcionar correctamente. Mahout podría ser lento incluso.
100K – 1M	Mahout empieza a ser una buena elección. Su flexible API puede hacer que sea nuestra opción preferida, incluso aunque no haya una mejora de rendimiento.
1M – 10M	Mahout es una opción excelente en este rango.
> 10M	Mahout sobresale donde otros fallan.

Figura 12: Tabla métodos de clasificación.

La razón de que Mahout tenga ventaja con grandes conjuntos de datos es que como los datos de entrada aumentan, el tiempo o los requisitos de memoria para el entrenamiento puede que no aumenten linealmente en un sistema no escalable, sino exponencialmente por ejemplo.

Un sistema que se ralentiza el doble con el doble de datos puede ser aceptable pero si aumentan los datos 5 veces y el sistema se ralentiza como 100 veces más, entonces no. Habría que encontrar otra solución, aquí es donde Mahout vendría perfecto.

En general, los algoritmos de clasificación de Mahout tienen una necesidad de recursos que no aumenta tan rápido como los ejemplos de entrenamiento y en la mayoría de casos los recursos de computación pueden ser paralelizados.

Parece que Mahout es la herramienta perfecta para la puesta en producción de estos algoritmos.

1.3.1. Terminología de los clasificadores

Antes de empezar a hablar más técnicamente de los clasificadores vamos a repasar la terminología básica que rodea el entorno de un clasificador:

Modelo	Programa que toma decisiones; en clasificación la salida del algoritmo de entrenamiento es un modelo.
Datos de entrenamiento	Un subconjunto de ejemplos de entrenamiento etiquetados con el valor de la variable objetivo y usados como entrada del algoritmo de aprendizaje para producir el modelo.
Datos de prueba	Una porción de los datos de entrenamiento con el valor de la variable objetivo ocultos para que puedan ser usados para evaluar el modelo.
Entrenamiento	El proceso de aprendizaje que usa los datos de entrenamiento para producir un modelo. Este modelo puede estimar la variable objetivo dada por las variables predictoras como entrada.
Ejemplo de entrenamiento	Una entidad con características que serán usadas como entrada para el algoritmo de aprendizaje.
Característica	Una característica conocida de un entrenamiento o nuevo ejemplo.
Variable	En este contexto, el valor de una característica o una función de varias características. Es algo diferente al uso de una variable en un programa.
Registro	Un contenedor donde se almacena un ejemplo. Está compuesto de campos.
Campo	Parte de un registro que contiene el valor de una característica (variable).
Variable predictora	Una característica seleccionada para usarla como entrada al modelo de clasificación. No todas las características necesitan ser usadas. Algunas pueden ser combinaciones algorítmicas de otras.
Variable objetivo	Una característica que el modelo de clasificación intenta estimar, es categórica y determinarla lo que busca el sistema de clasificación.

Figura 13: Terminología.

Aquí podemos ver una imagen que nos ayuda a ver donde participa cada uno los elementos de arriba:

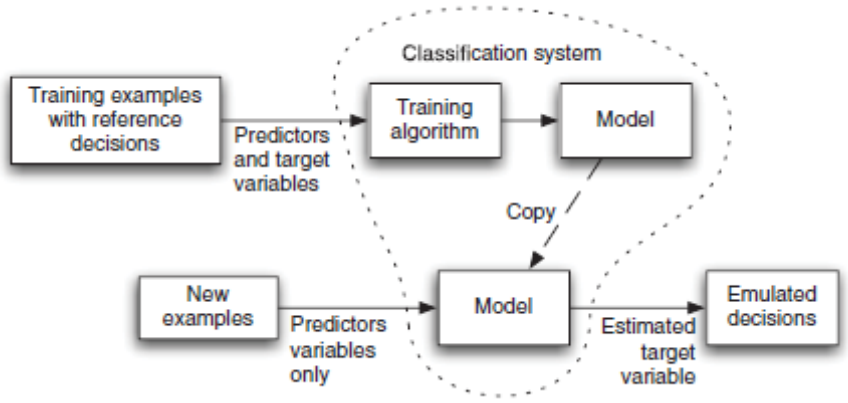


Figura 14: Flujo clasificación Ref. [27]

Vemos en la figura como introducimos primero los ejemplos de entrenamiento con las variables predictoras al algoritmo de aprendizaje, con esta información el algoritmo crea el modelo al que le podemos pasar los nuevos ejemplos que no tienen la variable objetivo. El

modelo calcularía cual sería el valor de dicha variable a partir de la información recogida de los ejemplos de entrenamiento.

1.3.3. Tipos de las variables predictoras

Las variables de los datos de entrada pueden ser de varios tipos: numéricos, palabras, texto o un valor sobre un conjunto definido. En la siguiente tabla se ve sobre qué tipo de datos trabaja cada variable:

Tipo	Descripción
Continua	Es un valor en punto flotante. Podría ser un precio, una talla, tiempo o algo más que tenga una magnitud numérica.
Categórica	Puede tener un valor dentro de un conjunto especificado de valores. Normalmente se trata de un conjunto reducido de valores, como mínimo de dos, aunque a veces puede ser bastante grande. Los booleanos son generalmente tratados como categóricos, otro ejemplo puede ser un ID de vendedor.
Palabra	Es como un valor categórico pero tiene un conjunto abierto de valores.
Texto	Es una secuencia de valores "palabra", todos del mismo tipo. Un texto es un claro ejemplo de estos valores, pero una lista de emails o URLs puede serlo también.

Figura 15: Tipos variables.

Hay que tener cuidado de no confundir valores continuos y categóricos, por ejemplo es común que se trate números de identificación como continuos cuando son categóricos ya que son un conjunto de valores ya especificados.

Tipos usados por algunos algoritmos:

SGD y random forest → Continuas.
Naive Bayes → No puede usar continuas.

1.3.4. Flujo de trabajo de un clasificador

El proceso de clasificación tiene varias etapas, vamos a enumerarlas para hacer más fácil la comprensión de cómo funciona un clasificador:

1. **Entrenar al modelo.**

    1.1. Definir variable objetivo.
    1.2. Colectar datos históricos.
    1.3. Definir variables predictoras.
    1.4. Seleccionar algoritmo de aprendizaje.
    1.5. Usar el algoritmo para entrenar al modelo.

2. **Evaluar el modelo.**

    2.1. Ejecutar los datos de prueba.
    2.2. Ajustar la entrada (usar diferentes predictores, algoritmos o ambos).

3. **Usar el modelo en producción.**

    3.1. Introducir nuevos ejemplos para estimar variables objetivo desconocidas.
    3.2. Reentrenar el modelo lo que sea necesario.

1.3.5. Tratamiento de los datos antes de usar el algoritmo de aprendizaje

Antes de introducir los datos para que los trate el algoritmo hay que prepararlos, partimos de datos sin formatear adecuadamente, por ejemplo un csv, obtenemos los ejemplos de entrenamiento con las variables objetivos en un formato adecuado para el siguiente paso que sería separar cada ejemplo en tokens para que sean transformados a vectores, aquí pasarían a ser un valor numérico identificativo del valor de cada variable. Una vez hecho todo esto podemos trabajar con el algoritmo de aprendizaje. En la siguiente imagen podemos ver el proceso:

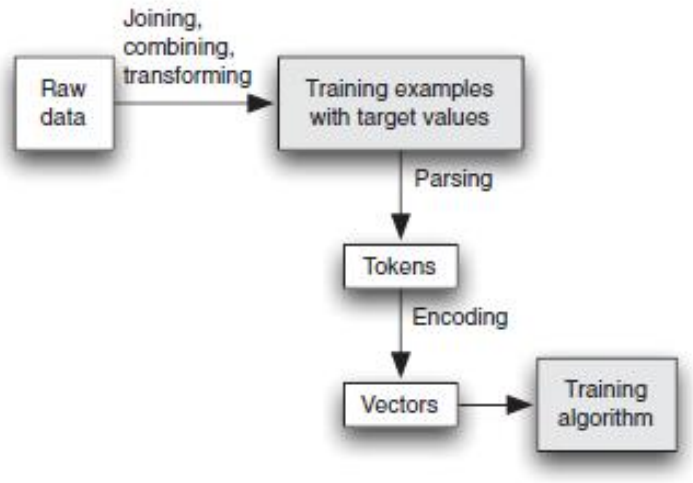


Figura 16: Clasificador Ref. [27]

A la hora de transformar los datos de entrada a vectores tenemos varias formas según el algoritmo o el método a utilizar. En la siguiente tabla podemos ver los diferentes métodos:

Método	Beneficio	Coste	Dónde se usa
Usar una celda de vector por valor categórico, continuo o palabra	No hay colisiones y reversible fácilmente.	Requiere dos pasos (uno para asignar celdas y otro para los valores) y vectores pueden tener diferentes tamaños.	Al volcar un índice Lucene como vectores para clustering.
Representar vectores implícitamente como paquetes de palabras	Un paso y no hay colisiones.	Hace difícil usar primitivas de álgebra lineal y representar valores continuos. Hay que formatear los datos en un formato no-vectorial especial.	Bayesiano ingenuo (Naive Bayes)
Utilizar hashing	Un paso, el tamaño del vector es fijado antes. Buena opción para primitivas de algebra lineal.	Requiere tratar las colisiones y la interpretación del modelo puede ser compliada.	En *Online Logistic Regression* y otros aprendices SGD

Figura 17: Tabla métodos.

1.3.6. Algoritmos de aprendizaje utilizados por Mahout

A continuación podemos ver una tabla con los algoritmos utilizados por Mahout, así como sus características principales y el modelo de ejecución llevado a cabo:

Tamaño del conjunto de datos	Algoritmo	Modelo de ejecución	Características
Pequeño a medio (Menos de decenas de millones de ejemplos)	SGD: Stochastic gradient descent (Descenso por gradiente estocástico)	Secuencial, online, incremental	Usa todo tipo de variables predictoras; Eficiente sobre el rango apropiado de datos.
	SVM: Support vector machine (Máquina de soporte vectorial)	Secuencial	Todavía experimental; Eficiente sobre el rango apropiado de datos.
Medio a grande (De miles o cientos de millones de ejemplos)	Naive Bayes (Bayesiano ingenuo)	Paralelo	Prefiere datos con valores "texto"; Efectivo y útil para los conjuntos de datos demasiado grandes para SGD o SVM.
	Complementary Naive Bayes (Bayesiano ingenuo complementario)	Paralelo	Algo más costoso de entrenar que el Naive Bayes; Tiene características parecidas al Naive Bayes.
Pequeño a medio (Menos de decenas de millones de ejemplos)	Random Forests (Bosques aleatorios)	Paralelo	Usa todo tipo de variables predictoras; Alto tiempo de procesamiento para el entrenamiento; No se utiliza mucho (todavía); Costoso pero ofrece clasificaciones interesantes y complejas; Maneja relaciones no lineales y condicionales en datos mejor que otras técnicas.

Figura 18: Algoritmos Mahout.

En *easyMahout* están implementados los algoritmos de clasificación *Naive Bayes* y *Complementary Naive Bayes* para datos distribuidos. El algoritmo *SGD* también esta implementado parcialmente pero no está en producción todavía.

Recomendaciones sobre algunos algoritmos:

Naive Bayes → Recomendado para +10M de ejemplos y variables *single text-like*.
SGD → Para otros casos se recomienda este algoritmo, otros están en proceso de desarrollo todavía.

2. Estado del arte

2.1. Big Data

El término Big Data hace referencia a una inmensa y compleja colección de datos (estructurados, no estructurados y semi-estructurados) también llamada *dataset*, la cual, debido a su gran tamaño y características, imposibilita su tratamiento por medio de los tradicionales sistemas de bases de datos y aplicaciones de procesamiento de datos comunes (aplicaciones estadísticas…).

Podemos describir Big Data mediante el modelo de las tres V: *volumen, velocidad y variedad*.

- *Volumen*: Muchos factores han contribuido a que el volumen de los datos haya incrementado exponencialmente en las últimas décadas. Una de las causas de este crecimiento se debe al almacenamiento de datos de ubicación procedentes de dispositivos móviles (ubicación GPS, sensores inalámbricos, VANETs), sistemas de teledetección de cámaras, micrófonos, lectores de radiofrecuencia… Estos datos no estructurados crecen muy rápidamente.

- *Variedad*: Actualmente los datos vienen en formatos de todo tipo y provienen de una amplia gama de fuentes y dispositivos. Desde datos estructurados en sistemas de BB.DD tradicionales a otros desestructurados tales como documentos de texto, redes sociales, emails, videos, imágenes, audios, datos numéricos científicos (áreas como la meteorología, la genética, la conectómica…) o sistemas de finanzas. Éstos son solo un pequeño ejemplo de la inmensa variedad de datos a los que nos referimos.

- *Velocidad*: Una vez que ya conocemos qué cantidad y variedad de datos necesitamos procesar, podemos comprender la velocidad de procesamiento que precisamos para obtener la información correcta en un determinado momento, trabajando en muchas ocasiones con torrentes de datos en tiempo real. Para conseguir tal velocidad, principalmente se trabaja en la nube, haciendo uso de los recursos exclusivamente necesarios permitiendo una significativa reducción de costes a lo que supondría poseer y mantener tal infraestructura.

Actualmente existen aproximadamente 2.000 millones de usuarios de internet, 4.600 millones de teléfonos móviles en 2011, Twitter procesa 7TB de datos cada día mientras que Facebook 10TB. Además, el 80% de los datos son no estructurados. Necesitamos la Big Data para comprender tal cantidad de datos.

Una estimación de IBM dice que en 2012 cada día fueron creados cerca de 2,5 trillones de bytes de datos, o lo que es lo mismo, aproximadamente el 90% de los datos que existen en el mundo, fueron creados durante sólo los dos últimos años.

Además, The McKinsey Global Institute estima que el volumen de datos está creciendo un 40% al año, y que se multiplicara por 44 entre 2009 y 2020.

Como ejemplo, la compañía telefónica China Mobile, que genera entre 5 y 8TB de datos cada día, consiguió procesar diez veces más de datos y reducir el gasto a la mitad usando esta nueva tecnología.

Si Big Data es analizada en combinación con los datos tradicionales de la empresa, una empresa puede llegar a comprender mejor su negocio, propiciando una mejor productividad, una mejora en innovación o ser más competitivas respecto a la competencia. Bajo ese punto de vista, podemos definir Big data como el conjunto de procesos, tecnologías y modelos de negocio que están basados en datos y en capturar el valor que los propios datos encierran, buscando el beneficio para la empresa. Además, los modelos de negocio de redes sociales como Twitter o LinkedIn se basan en la captura y el análisis de todos los datos referentes a los usuarios.

2.1.1. CLOUDERA

Cloudera es una de las compañías que más está aprovechando el gran crecimiento e interés del Big Data. Basando su negocio en Hadoop, software 100% open-source, ofrece soluciones gratuitas y de pago para explotar el Big Data.

Como cabe imaginarse, para explotar tal cantidad de datos se necesitaría una gran infraestructura inaccesible para la mayoría de personas y empresas, y es aquí donde entra Cloudera. Sus ingresos provienen de la venta de estos servicios y del soporte técnico ofrecido a los clientes. Además, gran parte de sus beneficios son donados a varios proyectos open source de licencia Apache, tales como Apache Hive, Apache Avro, Apache HBase, etc., así como sponsor de Apache Software Fundation.

2.2. Hadoop: Framework para aplicaciones distribuidas

Figura 19: Logo Hadoop. Ref. [7]

Hadoop aparece como una solución al problema planteado por el *Big Data*, una forma de tratar volúmenes masivos de datos de forma eficiente y rápida. Hadoop es un framework *open source* para ejecutar aplicaciones distribuidas y permite trabajar con miles de nodos y petabytes de datos. Inspirado en los documentos Google para *MapReduce* y *Google File System* (GFS), se trata de un proyecto de Apache que sigue en fase de desarrollo actualmente.

Yahoo! ha sido el mayor contribuyente al proyecto, usándolo actualmente en su negocio. Hadoop fue creado por Doug Cutting, que lo llamó así por el elefante de peluche de su hijo, y fue desarrollado originalmente para apoyar la distribución del proyecto de motor de búsqueda *Nutch*.

Hadoop nació en un paper escrito en 2004 por Doug Cutting, ingeniero de Google por aquellos tiempos, en el que describía las técnicas para dividir un problema sobre datos voluminosos en sucesivos subproblemas, distribuirlos entre decenas o cientos de nodos, y luego combinarlos en un conjunto reducido y fácil de analizar. Disconforme con ciertos condicionamientos de Google, acabó por marcharse a Yahoo, donde continuó con su desarrollo hasta concretarlo en 2008. Actualmente trabaja como arquitecto de software de Cloudera, cuya distribución de Hadoop lidera el mercado y ha reclutado expertos de casi todo el espectro del Silicon Valley: el CEO viene de Oracle, el *chief scientist* de Facebook y el CTO de Yahoo. Cloudera ofrece su distribución de Hadoop, y el gestor básico, de forma gratuita para *clusters* de hasta 50 máquinas.

La era comercial ha comenzado en 2011. Los tres grandes proveedores de bases de datos (Oracle, IBM, Microsoft) ya han adoptado Hadoop. A menudo se ha definido a Hadoop como una tecnología para el tratamiento de datos no estructurados, Cutting corrige esta idea: la combinación de escalabilidad, flexibilidad y bajo coste, hace que pueda aplicarse a todo tipo de datos.

Como curiosidad, El 19 de febrero de 2008, Yahoo! lanzó lo que era la más grande aplicación de producción Hadoop. El Yahoo! Search Webmap es una aplicación de Hadoop que se ejecuta en más de 10.000 núcleos Linux y produce datos que se utilizan actualmente en todos los resultados de búsqueda de Yahoo!. En junio de 2009, Yahoo! hizo disponible el código fuente de la versión de Hadoop que usa en producción. Aparte de Yahoo!, otras organizaciones usan Hadoop para ejecutar cómputos enormes distribuidos. Algunas de estas empresas son:

- eBay
- Fox Interactive Media
- IBM
- LinkedIn
- PowerSet (ahora parte de Microsoft)
- Tuenti
- Twitter
- 1&1
- AOL
- Metaweb (ahora parte de Google)
- Veoh

Hadoop es capaz de almacenar toda clase de datos: estructurados, no estructurados, semiestructurados, archivos de registro, imágenes, video, audio...

Desarrollado en Java es un sistema distribuido que usa una arquitectura Master-Slave, para almacenar su Hadoop Distributed File System (HDFS) y algoritmos de MapReduce para hacer cálculos.

2.2.1. Arquitectura

Hadoop consiste básicamente en el Hadoop Common, que proporciona acceso a los sistemas de archivos soportados por Hadoop. El paquete de software de Hadoop Common contiene los archivos .jar y los scripts necesarios para hacer correr Hadoop. El paquete también proporciona código fuente, documentación, y una sección de contribución que incluye proyectos de la Comunidad Hadoop.

Una funcionalidad clave es que para la programación efectiva de trabajo, cada sistema de archivos debe conocer y proporcionar su ubicación: el nombre del rack (más precisamente, del switch) donde está el nodo Master. Las aplicaciones Hadoop pueden usar esta información para ejecutar trabajo en el nodo donde están los datos y, en su defecto, en el mismo rack/switch, reduciendo así el tráfico de red. El sistema de archivos HDFS usa esto cuando replica datos, para intentar conservar copias diferentes de los datos en racks diferentes. El objetivo es reducir el impacto de un corte de energía de rack o de otro fallo de modo que incluso si se producen estos eventos, los datos todavía puedan ser legibles.

Un clúster típico Hadoop incluye un nodo maestro y múltiples nodos esclavo. El nodo maestro Hadoop requiere tener instalados en los nodos del clúster JRE 1.6 o superior, y SSH.

- ***Rack***

En Hadoop se denomina *rack* a la combinación de “nodos de datos”. Un *rack* puede tener un máximo de 40 *nodos máster*. Cada rack tiene un switch que le permite comunicarse con los distintos *racks* del sistema, sus *nodos* y con los procesos clientes.

- ***Nodo Máster***

Consiste en jobtracker (rastreador de trabajo), tasktracker (rastreador de tareas), namenode (nodo de nombres), y datanode (nodo de datos). Es el encargado de almacenar los metadatos asociados a sus *nodos slave* dentro del *rack* del que forma parte.

El nodo máster es el responsable de mantener el estatus de sus *nodos slave*, estableciendo uno de ellos como *nodo pasivo*, que se convertirá en nodo máster, si por cualquier motivo éste se quedara bloqueado. Uno de los problemas que tiene Hadoop es que a veces el *nodo pasivo* no está sincronizado con el *nodo máster* original, al asumir las funciones de éste dentro del proceso.

- ***Nodo slave (compute node)***

Consiste en un nodo de datos y rastreador de tareas. Es el nodo encargado de almacenar la información que se está procesando por el nodo máster en un momento concreto.

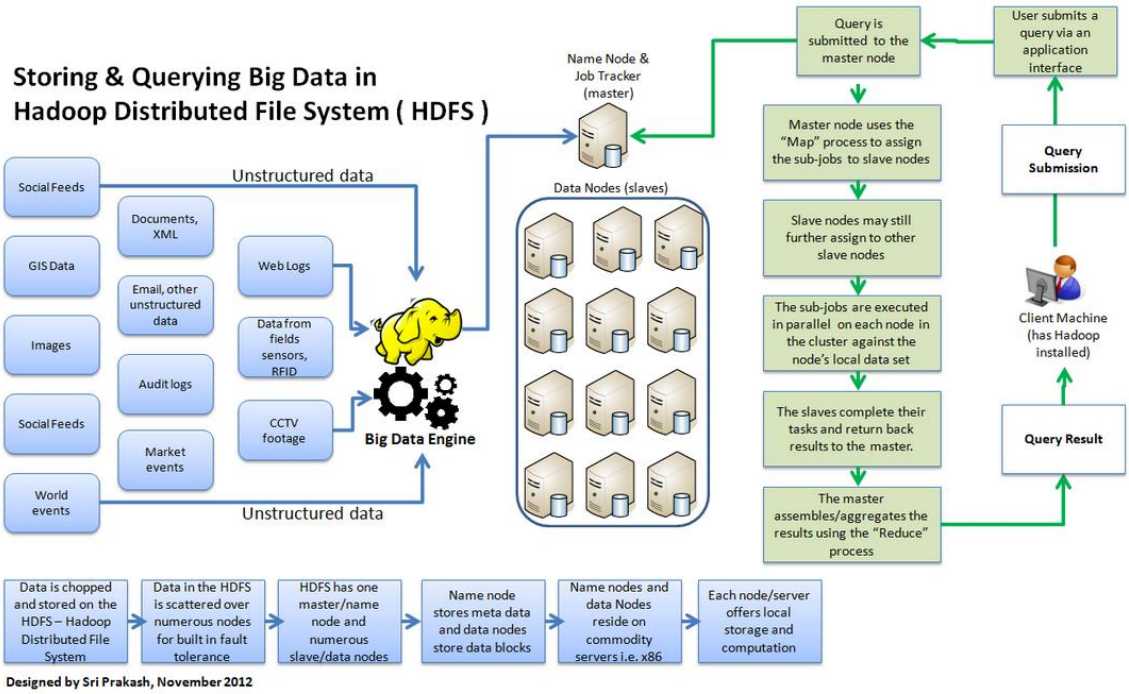


Figura 20: Big Data y HDFS. Ref. [30]

2.2.2. Sistemas de Archivos

- HDFS: El sistema de ficheros propio de Hadoop. Está diseñado para la escala de decenas de petabytes de almacenamiento y funciona sobre los sistemas de archivos base.
- Amazon S3: Éste se dirige a clústeres almacenados en la infraestructura del servidor bajo la demanda de Amazon Elastic Compute Cloud. No hay conciencia de racks en este sistema de archivos, porque es remoto.

- CloudStore (antes Kosmos Distributed File System), el cual es consciente de los racks.
- FTP: éste almacena todos sus datos en un servidor FTP accesible remotamente.
- HTTP y HTTPS de solo lectura.

Hadoop puede trabajar directamente con cualquier sistema de archivos distribuido, el cual puede ser montado por el sistema operativo subyacente simplemente usando la URL file://, sin embargo esto tiene un precio: la pérdida de la localización. Para reducir el tráfico de red Hadoop necesita saber qué servidores están más próximos a los datos. Esto incluye Amazon S3, y el almacén de archivos CloudStore, a través de las URLs *s3://* y *kfs://*.

Varios puentes de sistemas de archivos de terceros han sido escritos, ninguno de los cuales están actualmente en las distribuciones de Hadoop. Estos pueden ser de propósito más general que HDFS, el cual está orientado hacia grandes archivos y solo ofrece un subconjunto de la semántica esperada de sistema de archivos Posix.

El sistema de archivos Hadoop (HDFS) es un sistema de archivos distribuido diseñado para ejecutarse en hardware. Tiene muchas similitudes con otros sistemas distribuidos existentes, sin embargo, las diferencias con respecto a ellos son significativas.

HDFS es altamente tolerante a fallos y está diseñado para ser implementado en hardware de bajo coste. HDFS proporciona acceso de alto rendimiento para datos de aplicación y es adecuado para las aplicaciones que tienen grandes conjuntos de datos. HDFS relaja unos requisitos de POSIX para permitir el acceso de streaming para presentar los datos del sistema. HDFS fue construido originalmente como la infraestructura para el proyecto de motor de búsqueda Apache Nutch web. HDFS es ahora un subproyecto de Apache Hadoop. La URL del proyecto es http:// hadoop.apache.org/hdfs/

HDFS tiene una arquitectura maestro-esclavo. Un clúster HDFS consta de un solo *NameNode*, un servidor maestro que administra el espacio de nombres del sistema de archivo y regula el acceso a los archivos por parte de los clientes. Además, hay un número de *DataNodes*, generalmente uno por cada nodo del clúster, que administra el almacenamiento de información conectado a los nodos sobre los que corren. HDFS expone un espacio de nombres del sistema de archivo y permite que los datos de usuario se almacenen en archivos.

Internamente, un archivo se divide en uno o más bloques y estos bloques se almacenan en un conjunto de *DataNodes*. El *NameNode* ejecuta operaciones de espacio de nombres del sistema archivos como abrir, cerrar, renombrar archivos y directorios. También determina la asignación de bloques para *DataNodes*. Los *DataNodes* son responsables de servir las peticiones para leer y escribir desde el archivo de clientes del sistema.

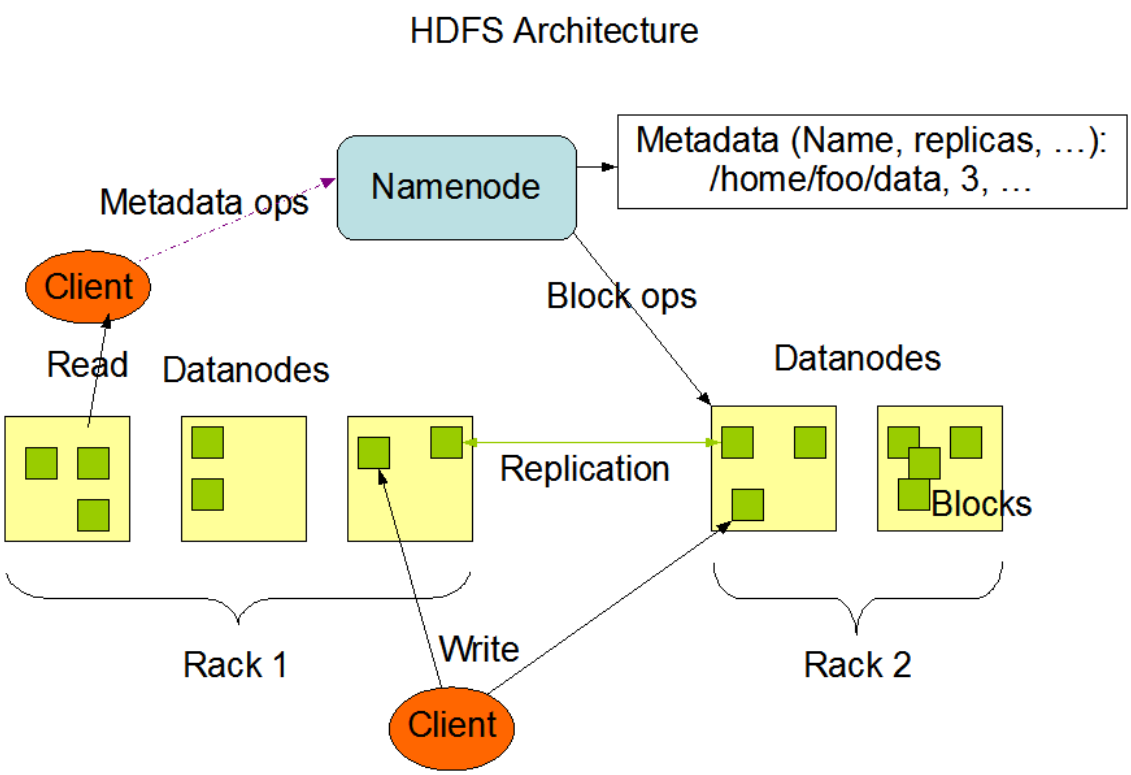


Figura 21: Arquitectura HDFS. Ref. [13]

NameNode y *DataNode* son piezas de software diseñadas para ejecutarse en máquinas de productos básicos. Estas máquinas suelen correr un sistema operativo GNU / Linux (OS). HDFS está construido utilizando el lenguaje Java, y cualquier equipo que admita Java puede ejecutar el *NameNode* o el software *DataNode*.

El uso del lenguaje Java altamente portátil significa que HDFS se puede implementar en una amplia gama de máquinas. Una implementación típica tiene una máquina dedicada que ejecuta sólo el software *NameNode*. Cada uno de los otros equipos del clúster ejecuta una instancia del software *DataNode*. La arquitectura no excluye la ejecución de múltiples *DataNodes* en la misma máquina, pero en una implementación real rara vez es el caso.

La existencia de un único *NameNode* en un clúster simplifica en gran medida la arquitectura del sistema. El *NameNode* es el árbitro y el repositorio de todos los metadatos HDFS. El sistema está diseñado de tal manera que datos del usuario nunca fluyen a través de la *NameNode*.

2.2.3. Replicación de datos

HDFS está diseñado para almacenar con fiabilidad archivos muy grandes a través de máquinas en un gran grupo. Almacena cada archivo como una secuencia de bloques; todos los bloques de un archivo, excepto el último bloque son del mismo tamaño. Los bloques de un archivo se replican para tolerancia a fallos. El tamaño del bloque y el factor de replicación son configurables por archivo. Una aplicación puede especificar el número de réplicas de un archivo. El factor de replicación se puede especificar en el momento de creación de archivos y

se puede cambiar más adelante. Los archivos en HDFS son de una sola escritura y tienen estrictamente un escritor en cualquier momento.

El *NameNode* toma todas las decisiones con respecto a la replicación de bloques. Recibe periódicamente un Heartbeat (pulso) y un Blockreport (informe de bloque) de cada uno de los *DataNodes* del clúster. La recepción de un latido del corazón implica que el *DataNode* funciona correctamente. Un Blockreport contiene una lista de todos los bloques en un *DataNode*.

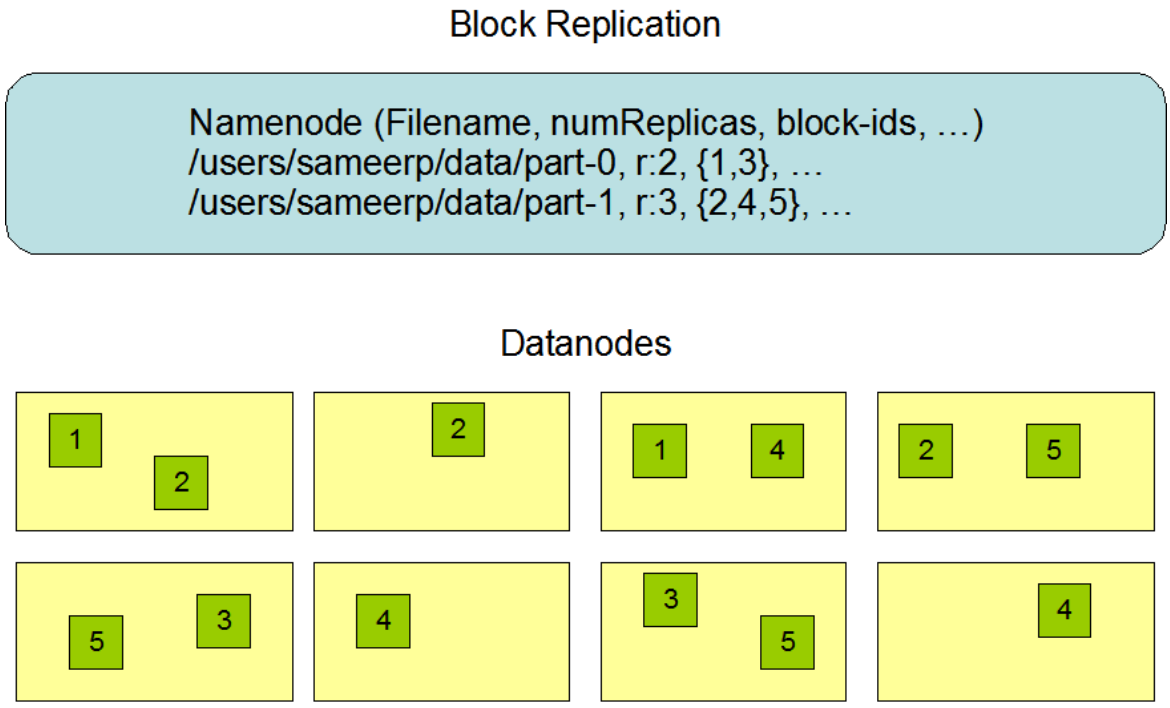


Figura 22: Replicación de datos. Ref. [13]

2.2.4. La persistencia de los metadatos del sistema de archivos

El espacio de nombres HDFS es almacenado por el NameNode. El NameNode utiliza un registro de transacciones llamado EditLog para grabar persistentemente cada cambio que se produce para presentar los metadatos del sistema. Por ejemplo, la creación de un nuevo archivo en HDFS hace que el NameNode inserte un registro en la EditLog indicando esto. Del mismo modo, cambiar el factor de replicación de un archivo hace que un nuevo registro se inserte en el EditLog.

El NameNode utiliza un archivo en su sistema de archivos del sistema operativo del servidor local para almacenar el EditLog. Todo el espacio de nombres del sistema de archivos, incluyendo la asignación de bloques de los archivos y las propiedades del sistema de archivos, se almacenan en un archivo llamado FsImage. El FsImage se guarda como un archivo en el sistema de archivos local del NameNode también.

El NameNode mantiene una imagen de todo el espacio de nombres del sistema de archivos y el archivo BLOCKMAP en la memoria. Este elemento clave de metadatos está diseñado para ser compacto, de modo que una NameNode con 4 GB de RAM es suficiente para apoyar un

gran número de archivos y directorios. Cuando el NameNode arranca, lee el FsImage y EditLog desde el disco, se aplica todas las transacciones de la EditLog a la representación en memoria del FsImage y vuelca esta nueva versión en una nueva FsImage en el disco. A continuación, se puede truncar el viejo EditLog debido a que sus operaciones se han aplicado a la FsImage persistente. Este proceso se denomina un punto de control. En la implementación actual, un puesto de control sólo se produce cuando la NameNode se pone en marcha. Se está trabajando para apoyar puntos de control periódico en un futuro próximo.

Los DataNode almacenan los archivos HDFS en su sistema de archivos local. El DataNode no tiene conocimiento acerca de los archivos HDFS. Almacena cada bloque de datos HDFS en un archivo aparte en su sistema de archivos local. El DataNode no crea todos los archivos en el mismo directorio. En su lugar, se utiliza una heurística para determinar el número óptimo de archivos por directorio y crea subdirectorios apropiadamente. No es óptimo para crear todos los archivos locales en el mismo directorio ya que el sistema de archivos local podría no ser capaz de soportar de manera eficiente un gran número de archivos en un solo directorio. Cuando un DataNode inicia, explora a través de su sistema de archivos local, genera una lista de todos los bloques de datos HDFS que corresponden a cada uno de estos archivos locales y le envía un informe a la NameNode: este es el Blockreport.

2.2.5. Los protocolos de comunicación

Todos los protocolos de comunicación HDFS se colocan en capas en la parte superior del protocolo TCP / IP. Un cliente establece una conexión a un puerto TCP configurable en la máquina NameNode. Habla el ClientProtocol con el NameNode. Los DataNode hablan con el NameNode mediante el protocolo DataNode. Una llamada a procedimiento remoto (RPC) de la abstracción envuelve tanto el Protocolo de Cliente y el Protocolo DataNode. Por diseño, el NameNode nunca inicia las RPC. En cambio, sólo responde a las peticiones RPC emitidos por DataNodes o clientes.

2.2.6. Organización de datos

HDFS está diseñado para soportar los archivos de gran tamaño. Las aplicaciones que sean compatibles con HDFS son aquellos que se ocupan de grandes conjuntos de datos. Estas aplicaciones escriben sus datos sólo una vez, pero lo leen una o más veces y solicitan que éstas se lean en conformidad con la velocidad de transmisión. HDFS soporta una sola escritura y muchas lecturas semánticas en los archivos. Un tamaño de bloque típico utilizado por HDFS es de 64 MB. Por lo tanto, un archivo de HDFS está cortado en trozos de 64 MB, y si es posible, cada trozo residirá en un DataNode diferente.

2.2.7. Puesta en escena

La petición del cliente para crear un archivo no llega a la NameNode inmediatamente. De hecho, en un principio el cliente HDFS almacena en caché los datos del archivo en un archivo local temporal. La aplicación le redirige, de manera transparente, a este archivo local temporal. El NameNode inserta el nombre del archivo en la jerarquía del sistema de archivos y asigna un bloque de datos para ello. El NameNode responde a la solicitud del cliente con la identidad de la DataNode y el bloque de datos de destino. Cuando se cierra un archivo, los

datos que quedan en el archivo local temporal se transfiere al DataNode. El cliente le dice al NameNode que el archivo está cerrado. En este punto, el NameNode compromete la operación de creación de archivos en un almacén persistente. Si el NameNode muere antes de cerrar el archivo, el archivo se pierde. El enfoque anterior se ha adoptado después de una cuidadosa consideración de las aplicaciones de destino que se ejecutan en HDFS. Estas aplicaciones necesitan streaming de escrituras en los archivos. Si un cliente escribe en un archivo remoto directamente sin ningún almacenamiento en búfer en el lado del cliente, la velocidad de la red y la congestión en la red impacta en el rendimiento considerablemente. Este enfoque no carece de precedentes. A principios de los sistemas de archivos distribuidos, por ejemplo, AFS, se han utilizado en el cliente de caché para mejorar el rendimiento. Un requisito POSIX se ha relajado para lograr un mayor rendimiento de los archivos de datos.

2.2.8. Canalización de replicación

Cuando un cliente está escribiendo los datos en un archivo de HDFS, sus datos se escriben primero en un archivo local, como se explica en la sección anterior. Supongamos que el archivo HDFS tiene un factor de replicación de tres. Cuando el archivo local acumula un bloque completo de datos de usuario, el cliente recupera una lista de DataNodes del NameNode. Esta lista contiene los DataNodes que serán sede de una réplica de ese bloque. A continuación, el cliente vacía el bloque de datos a la primera DataNode. La primera DataNode inicia la recepción de los datos en pequeñas porciones (4 KB), escribe cada porción de su depósito local y las transferencias de esa porción de la segunda DataNode en la lista. El segundo DataNode, a su vez inicia la recepción de cada porción del bloque de datos, escribe que parte a su repositorio, y luego vuelca esa porción de la tercera DataNode. Por último, la tercera DataNode escribe los datos a su repositorio local. Por lo tanto, un DataNode puede estar recibiendo datos de la anterior en la tubería y en los mismos datos de reenvío de tiempo a la siguiente en la tubería. Por lo tanto, los datos se canalizan de un DataNode a la siguiente.

2.2.9. Accesibilidad

HDFS se puede acceder desde las aplicaciones de muchas maneras diferentes. Nativa, HDFS proporciona una API Java para aplicaciones que utilizan. AC envoltorio idioma API Java también está disponible. Además, un navegador HTTP también se puede utilizar para navegar por los archivos de una instancia HDFS. Se está trabajando para exponer HDFS a través del protocolo WebDAV.

2.2.10. FS Shell

HDFS permite que los datos de usuario se organizen en forma de archivos y directorios. Proporciona una interfaz de línea de comandos shell llamado FS que permite al usuario interactuar con los datos en HDFS. La sintaxis de este conjunto de comandos es similar a otros depósitos (por ejemplo, bash, csh) con los que los usuarios ya están familiarizados. Estas son algunas pares de muestras de acción / comando:

Action	Command
Create a directory named `/foodir`	`bin/hadoop dfs -mkdir /foodir`
Remove a directory named `/foodir`	`bin/hadoop dfs -rmr /foodir`
View the contents of a file named `/foodir/myfile.txt`	`bin/hadoop dfs -cat /foodir/myfile.txt`

Figura 23: Shell FS Action-Command.

La interfaz de comandos Shell FS está dirigida hacia aplicaciones que necesitan un lenguaje script para interactuar con los datos almacenados.

2.2.11. DFSAdmin

El conjunto de comandos DFSAdmin se utiliza para administrar un clúster HDFS. Estos son los comandos que se utilizan sólo por un administrador HDFS. Estas son algunas muestras de acción / comando:

Action	Command
Put the cluster in Safemode	`bin/hadoop dfsadmin -safemode enter`
Generate a list of DataNodes	`bin/hadoop dfsadmin -report`
Recommission or decommission DataNode(s)	`bin/hadoop dfsadmin -refreshNodes`

Figura 24: DFSAdmin.

2.2.12. Interfaz del navegador

Una típica instalación HDFS configura un servidor web para exponer el espacio de nombres HDFS a través de un puerto TCP configurable. Esto permite al usuario navegar por el espacio de nombres HDFS y ver el contenido de sus archivos a través de un navegador web.

2.3. Mahout: Máquina de aprendizaje escalable y minería de datos

Figura 25: Mahout, máquina de aprendizaje escalable y minería de datos. Ref. [28]

Apache Mahout es un proyecto de Apache para producir implementaciones gratuitas de algoritmos de aprendizaje basados en máquinas distribuidas, o de otra manera escalable centrada básicamente en áreas de filtrado colaborativo, clustering y clasificación, aprovechando a menudo, y no limitándose a la plataforma Hadoop. Mahout también proporciona librerías Java para matemáticas comunes (algebra lineal y estadística), métodos y tipos de datos primitivos de Java. Mahout es un trabajo en proceso; el número de algoritmos implementados ha crecido sustancialmente pero todavía hay algoritmos que faltan.

Los principales algoritmos de Machine Learning implementados en Mahout son los siguientes:

Filtro Colaborativo	Breve descripción	Caso de uso	Local	MR
FC basado en usuario	Sistema de recomendación basado en la similitud entre usuarios y sus valoraciones.	Sistemas de recomendación de tiendas online.	x	
FC basado en objeto	Sistema de recomendación basado en la similitud entre objetos y sus valoraciones.	Sistemas de recomendación de tiendas online.	x	x
Factorización de matrices ALS	Factorización de matrices que soluciona los distintos problemas que presenta la factorización SVD.	Probado empíricamente sobre BBDD de votaciones de películas. (Datos explícitos)	x	x
Factorización de matrices SVD	Diseñado para reducir el ruido en matrices grandes, haciendo con esto que sean más pequeñas y que sea más fácil trabajar con ellas	Como precursor del almacenamiento en clúster, los sistemas de recomendación y la clasificación para realizar selección de recursos automáticamente	x	x
Factorización de matrices SVD++	Modificación del algoritmo SVD original llevada a cabo por Yehuda Koren.	Datos de valoración implícitos	x	

Clustering	Breve descripcion	Caso de uso	Local	MR
Canopy	Algoritmo de clustering que destaca por el cálculo de los canopies que formarán los clusters de salida.	Datos numéricos	x	x
K-Means	Algoritmo de clustering cuya principal característica es la elección del centro de cada cluster e iterando con los datos de entrada sobre ese.	Datos no numéricos	x	x
Fuzzy K-Means	Mejora del algoritmo K-Means permitiendo a los datos que pertenezcan a uno u otro cluster dependiendo del parámetro de fuzificación.	Datos no numéricos	x	x

Clasificación	Breve descripcion	Caso de uso	Local	MR
Regresión Logística	Clasificador brillante, rápido, simple y secuencial, capaz de aprendizaje on-line en entornos exigentes.	Recomiende publicidad a los usuarios, clasifique texto en categorías.	x	
Naive Bayes (y Complementario)				x
Random Forest				x
Hidden Markov Models	Implementaciones secuenciales y paralelas del algoritmo clásico de clasificación diseñado para modelar procesos del mundo real cuando el proceso de generación subyacente es desconocido.	Etiquetado de texto parte-del-discurso; reconocimiento del discurso.	x	
Perceptrón multicapa			x	

Figura 26: Algoritmos Mahout.

Mahout también tiene un gran número de algoritmos matemáticos de bajo nivel que los usuarios pueden encontrar útiles, pensados especialmente para la manipulación de Vectores (Matrices) con datos en gran escala.

2.3.1. Escalando Mahout en la nube

Escalar Mahout a una gran cantidad de datos de una manera eficaz no es tan fácil como simplemente añadir más nodos a un clúster Hadoop. Factores como la elección de algoritmo, el número de nodos, la selección de recursos y la escasez de datos (así como la memoria, el ancho de banda y la velocidad de los procesadores) juegan un papel importante al momento de determinar como de efectivo se puede escalar Mahout.

2.4. PIG

Pig: Lenguaje procedimental de alto nivel que permite la consulta de grandes conjuntos de datos semiestructurados utilizando Hadoop y la plataforma MapReduce.

Apache Pig es una plataforma para el análisis de grandes conjuntos de datos que consta de un lenguaje de alto nivel para expresar programas de análisis, junto con la infraestructura para la evaluación de los mismos. La característica sobresaliente de los programas de Pig es que su estructura es susceptible a la paralelización, lo que a su vez le permite manejar enormes cantidades de información. La capa de infraestructura de Pig se compone de un compilador que produce secuencias MapReduce, lo que permite a que los usuarios de Hadoop se enfoquen más en analizar los datos y dedicar menos tiempo en desarrollar aplicaciones MapReduce. El lenguaje de programación que utiliza Pig, Pig Latin, crea estructuras tipo SQL (SQL-like), de manera que, en lugar de escribir aplicaciones separadas de MapReduce, se pueda crear un script de Pig Latin el cual es automáticamente paralelizado y distribuido a través de un clúster.

Originalmente desarrollado por Yahoo en el año 2006, Pig fue adoptado por la Apache Software Foundation a partir del año 2007; un año después obtuvieron la versión inicial como parte de un subproyecto de Apache Hadoop.

Para lograr un mejor entendimiento sobre el objetivo de la creación de Pig, el equipo de desarrollo decidió definir una serie de enunciados que resumen el proyecto, mediante una similitud con el nombre:

- *Pigs eat anything:* Al igual que cualquier cerdo que come cualquier cosa, Pig puede operar con cualquier tipo de datos, sea éste estructurado, semi-estructurado o no estructurado.
- *Pigs live anywhere:* A pesar de que Pig fue inicialmente implementado en Hadoop, no está orientado solamente a esta plataforma. Su propósito es ser un lenguaje de procesamiento paralelo.
- *Pigs are domestic animals:* Pig está diseñado para ser controlado y modificado fácilmente por sus usuarios. Pig puede enriquecerse a través de funciones definidas por el usuario (UDF). Con el uso de UDFs se puede extender Pig para un procesamiento personalizado.
- *Pigs Fly:* Pig procesa datos rápidamente. La intención es mejorar el rendimiento y no las características, lo que evita que demasiada funcionalidad le impida "volar".

Naturalmente Yahoo!, al ser creador de Pig, fue el primer usuario de la plataforma, tanto para los procesos de búsqueda web como al incorporarlo en Hadoop. De hecho, más de la mitad de procesos que son ejecutados en Hadoop están basados en scripts de Pig Latin. Pero no sólo Yahoo ha utilizado Pig; a partir del año 2009 otras compañías comenzaron a adoptar Pig dentro de su procesamiento de datos, algunas de ellas son:

- *"Gente que podrías conocer"*, este componente de LinkedIn utiliza Hadoop y Pig para ofrecer recomendaciones de conocidos, páginas y empleos de interés.

- *"select count(*) from tweets"*, definitivamente SQL no es la opción para analizar tweets, retweets, usuarios, seguidores,etc., que conforman más de 12TB de información diaria. Twitter utiliza Pig para procesar estos logs de datos.
- AOL y WhitePages utilizan Pig para filtrar registros en sus procesos de búsqueda de información.

2.4.1. Pig Latin

Es un lenguaje de flujos de datos en paralelo. Esto es, que permite a los programadores describir como los datos provenientes de una o más entradas deben ser leídos, procesados y luego almacenados a uno o más flujos de salida en paralelo. La sintaxis de Pig Latin es muy similar a la de SQL, aunque Pig Latin es un lenguaje de transformación de datos y, por lo tanto, es similar a los optimizadores de consultas de base de datos de los sistemas de bases de datos actuales. Escribir programas MapReduce en Java puede consistir en más de cien líneas de código, según la complejidad de los mismos, mientras que los scripts de Pig Latin comúnmente no toman más de 10 líneas de código. En Pig Latin no existen condicionales tipo if o ciclos mediante el uso de for, dado que Pig Latin se enfoca en el flujo de los datos y no en describir el control del flujo de los datos como otros paradigmas de programación. Pig Latin utiliza operadores relacionales para efectuar diversas operaciones sobre los datos que se están analizando, desde la carga de los datos hasta su almacenamiento en un archivo. Aunque ya mencionaba que no existen ciclos for en Pig Latin, existe el operador *FOREACH* cuya naturaleza es iterar sobre las tuplas y transformar los datos para generar un nuevo conjunto de datos durante la iteración. Considerando la similitud de este operador con la terminología de base de datos, se podría decir que *FOREACH* es el operador de proyección de Pig. Algunos de los operadores relacionales existentes en Pig Latin se describen en la Tabla 1

Operador	Descripción
DISTINCT	Elimina duplicados en una relación.
FILTER	Selecciona un conjunto de tuplas de una relación basado en una condición
FOREACH	Itera las tuplas de una relación, generando un nuevo conjunto de datos
GROUP	Agrupa los datos en una o más relaciones
JOIN	Une dos o más relaciones (existe tanto inner join como outer join)
LIMIT	Establece el límite de tuplas de salida
LOAD	Carga datos de un sistema de archivos
ORDER	Ordena una relación basado en uno o más campos
SPLIT	Divide una relación en una o más relaciones
STORE	Almacena información transformada en el sistema de archivos

Figura 27: Operadores relacionales de PIG Latin.

Además de los operadores relacionales, existen a su vez operadores de diagnóstico que son de mucha utilidad para depurar los scripts de Pig Latin. El operador DUMP permite desplegar en pantalla el contenido de una relación, DESCRIBE imprime en pantalla el esquema detallado de una relación (campo y tipo) y, EXPLAIN permite visualizar cómo los operadores están agrupados en procesos MapReduce.

2.4.2. ¿No es suficiente con MapReduce? ¿Por qué otro lenguaje?

Una de las grandes ventajas que ofrece Pig es el uso de los operadores relacionales como *JOIN*, *FILTER*, *GROUP BY*, etc., los cuales en MapReduce resulta costosa la implementación, además de terminar con un algoritmo de cientos de líneas de código. En MapReduce no hay manera de optimizar o revisar el código del usuario para proveer estadísticas de desempeño, en cambio, Pig puede analizar los scripts de Pig Latin para revisión de errores y optimización del flujo de datos. Aunque en MapReduce el costo de escribir y mantener código es mucho mayor que en Pig o inclusive que en Jaql(Query Language for JSON), no siempre buscar una opción alterna a MapReduce puede ser lo más viable, puesto que es posible desarrollar algoritmos en MapReduce que no puedan ser tan fácilmente implementados en Pig o Jaql. De manera que para algoritmos no tan triviales y que sean sensibles a un muy alto rendimiento, MapReduce sigue siendo la mejor opción.

2.4.3. ¿Qué se puede resolver con Pig?

Así como el uso que varias empresas le han dado a Pig, existen muchos datos sin procesar (*raw data*) que constantemente están entregando información, de manera que combinar estos datos permite construir modelos de predicción del comportamiento. Pig podría ser utilizado para capturar todas las interacciones del usuario en un sitio web y dividir a los usuarios en varios segmentos. Así, para cada segmento se produce un modelo matemático que predice cómo los miembros de ese segmento responden a los tipos de anuncios o artículos de noticias. De tal modo que el sitio web pueda mostrar anuncios que sean más propensos a hacer clic o publicar noticias que tengan más probabilidades de atraer a los usuarios y hacer que regresen al sitio.

Si bien hemos visto hasta ahora que Pig permite crear procesos para analizar flujos de datos y con el uso de operadores relacionales hace aún más sencilla la agrupación, unión y agregación de los mismos, ¿Será que existe una similitud entre Pig y una herramienta ETL (*Extract Transform Load*)? En efecto, este es otro de los casos de uso de Pig. Dado que los datos son presentados en diversos formatos, necesitan ser procesados y almacenados en una base de datos para posteriormente ejecutar *queries* sobre ellos. Pig permite paralelización gracias a Hadoop y aunque diversas herramientas ETL permiten descomponer los procesos en pequeños segmentos, los *scripts* de Pig son aún más simples y fáciles de entender.

Sin embargo, esto no indica que sea un reemplazo de una herramienta de ETL, puesto que además de que no provee funcionalidades específicas de un ETL(y que no fue desarrollado para tal fin), el uso de Pig para todos los procesos de ETL sería una exageración cuando los datos razonablemente pueden manejarse en una instancia de base de datos.

2.4.4. Script en Pig Latin para calcular la temperatura máxima

```
smnData = LOAD '/SampleData/Pig/DF04_10M.TXT' USING PigStorage('\n');
cleanSmnData = FILTER smnData BY (chararray)$0 MATCHES
        '\\d{2}/\\d{2}/\\d{4}.*' and SIZE($0)>=66;
temperature = FOREACH cleanSmnData GENERATE SUBSTRING($0, 62, 66);
groupTemperature = GROUP temperature ALL;
maxTemperature = FOREACH groupTemperature GENERATE MAX(temperature.$0);
DUMP maxTemperature;
```

A continuación observamos en detalle cada uno de los pasos ejecutados por el script en Pig Latin:

smnData = LOAD '/SampleData/Pig/DF04_10M.TXT' USING PigStorage('\n');

Esta línea carga los datos del archivo DF04_10M.TXT (si sólo se hubiera indicado el directorio, leería todos los archivos contenidos en el mismo). PigStorage es la función por defecto para la carga de datos así que igualmente pudo haberse omitido, o bien haber indicado una función que sea más personalizada.

cleanSmnData = FILTER smnData BY (chararray)$0 MATCHES '\\d{2}/\\d{2}/\\d{4}.*' and SIZE($0)>=66;

Esta línea realiza el filtro (FILTER) del conjunto de datos de la relación smnData mediante el uso de una expresión regular (MATCHES) para crear un nuevo conjunto de datos que sólo contenga la información que interesa analizar, desde la fecha hasta la radiación solar, eliminando los encabezados y aquellas líneas que son menores a 66 caracteres, ya que pudieran haber presentado algún error desde la captura por las estaciones meteorológicas.

temperature = FOREACH cleanSmnData GENERATE SUBSTRING($0, 62, 66);

La línea anterior genera una transformación de las tuplas para obtener solamente los datos correspondientes a la columna TEMP.

groupTemperature = GROUP temperature ALL; maxTemperature = FOREACH groupTemperature GENERATE MAX(temperature.$0);

Estas líneas agrupan los valores transformados en la relación temperature y obtienen la temperatura máxima de dicha agrupación. Si pensáramos en SQL seguramente el valor máximo se podría obtener sin agrupar los valores, sin embargo, en Pig Latin la función MAX requiere que se haya ejecutado anteriormente la sentencia GROUP ALL para calcular valores máximos.

Si se quisieran optimizar las dos líneas de código anteriores, podemos utilizar esta opción, que nos permite obtener el mismo resultado pero de una manera más eficiente y rápida. Simplemente ordenamos la relación *temperature* de manera descendente y obtenemos el primer valor que sería la temperatura máxima. Este mismo tipo de "mejores prácticas" se utiliza también en SQL.

```
groupTemperature = ORDER temperature BY $0 DESC;
maxTemperature = LIMIT groupTemperature 1;

DUMP maxTemperature;
```

Y finalmente con el operador DUMP se imprime en pantalla la temperatura máxima registrada.

2.4.5. InfoSphere BigInsights

IBM InfoSphere BigInsights™ está basado en Apache Hadoop y está diseñado para ayudar a los profesionales de IT a comenzar a trabajar rápidamente con el análisis de grandes volúmenes de información mediante Hadoop. Facilita la instalación, integración y seguimiento de esta tecnología de código abierto.

InfoSphere BigInsights ofrece un gran valor a las organizaciones que se enfrentan a analizar volúmenes de datos a escala de Internet (petabytes) que existen en diversos formatos, y que pueden ser propagados a muchos lugares distintos; a empresas que están interesadas en una mayor flexibilidad para entender patrones y hacer análisis eficientes sobre cuestionamientos tipo "what if?".

La distribución de IBM de Apache Hadoop, InfoSphere BigInsights, contiene los siguientes componentes:

- *Jaql*, un lenguaje de consulta diseñado para JavaScript Object Notation (JSON), se utiliza principalmente para analizar datos semi-estructurados a gran escala.
- *Avro*, un sistema de serialización de datos.
- *Flume*, un servicio distribuido, confiable y de alta disponibilidad para mover de manera eficiente grandes cantidades de datos alrededor de un clúster.
- *HBase*, una base de datos distribuida no relacional escrita en Java.
- *Hive*, una infraestructura de *data warehouse* que facilita tanto la extracción de datos, transformación y carga (ETL), y el análisis de grandes conjuntos de datos que se almacenan en el sistema de archivos distribuidos Hadoop (HDFS).
- *Lucene*, un motor de búsqueda de texto de alto rendimiento escrito en Java.
- *Oozie*, un administrador de coordinación de flujos de trabajo.
- *Orchestrator*, un avanzado sistema de control de procesos MapReduce, el cual utiliza un formato JSON para describir los flujos de trabajo y las relaciones entre ellos.
- *Pig*, como se observó anteriormente, es una plataforma para el análisis de grandes conjuntos de datos, consiste en un lenguaje de alto nivel para la expresión de los programas de análisis y de una infraestructura para la evaluación de dichos programas.
- *BigInsights Scheduler*, se asegura que todos los procesos obtengan una adecuada asignación de recursos.
- *Zookeeper*, un servicio centralizado para el mantenimiento de la información de configuración, ofreciendo sincronización distribuida.

2.4.6. Conclusiones

Pig Latin puede utilizarse para crear algoritmos MapReduce de una manera más simple y sencilla, generando líneas de código mucho más legibles y mantenibles en el tiempo. Pig proporciona una abstracción útil sobre MapReduce; es decir, permite escribir sentencias de manipulación de datos y *queries* en un lenguaje de alto nivel y gracias a su modelo MapReduce subyacente es capaz de paralelizar automáticamente y escalar las operaciones realizadas proporcionando un fuerte apoyo para el trabajo con conjuntos de datos muy grandes.

Tanto Pig como Jaql ofrecen fácil programación para Hadoop, permitiendo un rápido desarrollo en comparación con la interfaz de Java que provee Hadoop.

Pig está más orientado hacia el enfoque de programación procedural y no hacia el enfoque declarativo (como es el estilo de SQL), además de proveer la capacidad de controlar los planes de ejecución. Si bien Pig ofrece un alto nivel de manipulación de datos primitivos como en la proyección y la unión, lo hace en un estilo mucho menos declarativo que SQL.

Quizás el ejemplo resulte sencillo como para poder observar el gran valor que ofrece Pig al analizar Big Data, sin embargo, recordemos que en un ambiente real se estarían involucrando otros tipos de contenido además de que el gran volumen de información generado (como puede ser el proveniente de estaciones meteorológicas) resulta demasiado complejo de analizar con otras herramientas tradicionales.

IBM InfoSphere BigInsights™ permite agilizar y facilitar la implementación de proyectos asociados con Big Data debido que además de incluir diversos subproyectos relacionados con Hadoop, también existe un gran número de componentes desarrollados por IBM como BigSheets, el cuál presenta una interfaz similar a una hoja de cálculo para que los usuarios puedan modelar, filtrar, combinar, explorar y graficar los datos obtenidos de diversas fuentes. Tras bambalinas, BigSheets traduce los comandos de usuario, expresados a través de una interfaz gráfica, en scripts de Pig ejecutados en un subconjunto de datos subyacentes. De esta manera, un analista puede explorar de manera iterativa diversas transformaciones eficientemente.

2.5. HIVE

2.5.1. Introducción

Figura 28: Logo HIVE. Ref. [15]

Hive es un sistema de almacenamiento de datos de Hadoop que facilita el resumen de datos fácilmente, consultas ad-hoc, y el análisis de grandes conjuntos de datos almacenados en los sistemas de archivos compatibles con Hadoop. Hive proporciona un mecanismo para la proyección de la estructura en estos datos y consultar los datos utilizando un lenguaje similar a SQL llamado HiveQL. Al mismo tiempo, este lenguaje también permite a los tradicionales programadores de map / reduce conectar sus mappers y reducers personalizados cuando es inconveniente o ineficiente para expresar esta lógica en HiveQL.

El software Apache Hive para el almacenamiento de datos facilita la consulta y la gestión de grandes conjuntos de datos que residen en el sistema de almacenamiento distribuido. Funciona sobre Apache Hadoop y nos ofrece:

- Herramientas que permitan de manera sencilla extraer, transformar o cargar los datos (ETL, Extract, Trasform, Load).
- Un mecanismo para estructurar los datos en una gran variedad de formatos.
- Acceso a los archivos almacenados, ya sea directamente en HDFS o en otros sistemas de almacenamiento de datos, como podría ser Apache HBase.
- Ejecutar consultas a través del paradigma MapReduce. Hive define un lenguaje simple de consulta similar a SQL, llamado QL, que permite a los usuarios familiarizados con SQL, consultar los datos de una manera similar. Al mismo tiempo, este lenguaje también permite a los programadores que están familiarizados con el marco MapReduce poder utilizar sus funciones MAP y sus funciones REDUCE personalizadas para realizar análisis más sofisticados que pueden no estar incorporadas en el lenguaje. QL también se puede ampliar con funciones personalizadas, agregaciones, y funciones de tabla. Hive no obliga a leer o escribir los datos en el formato "Hive ", Hive funciona igual de bien en Thrift, control delimitado, o los formatos de datos especializados. No está diseñado para cargas de trabajo OLTP y no ofrece consultas en tiempo real o actualizaciones a nivel de fila. Lo mejor es usarlo para trabajos por lotes en grandes conjuntos de datos anexados (como los weblogs).

Ejemplos como:

$ a=b
$ hive -e "describe $a"

Se están convirtiendo en algo bastante común. Esto es frustrante ya que Hive es estrechamente asociado con lenguajes de script. El tiempo de inicio de Hive de un par de segundos no es trivial cuando se hacen miles de manipulaciones tales como múltiples invocaciones hive-e. Las variables Hive combinan la capacidad de juego que conocemos, con cierta capacidad de sustitución limitada pero potente. Por ejemplo:

$ bin/hive -hiveconf a=b -e 'set a; set hiveconf:a; \

create table if not exists b (col int); describe ${hiveconf:a}'

Se convierte en:

```
Hive                                                                history
file=/tmp/edward/hive_job_log_edward_201011240906_1463048967.txt
a=b
hiveconf:a=b
OK
Time taken: 5.913 seconds
OK
col int
Time taken: 0.754 seconds
```

2.5.2. Pig y Hive

Apache Hive proporciona una capa de SQL en la parte superior de Hadoop. Toma consultas SQL y las traduce a trabajos MapReduce, casi de la misma forma que traduce Pig Pig Latin. Almacena datos en tablas y mantiene los metadatos relativos a las tablas, como las particiones y esquemas. Muchos ven Pig y Hive como competidores. Dado que ambos proporcionan una forma para que los usuarios operen sobre los datos almacenados en Hadoop sin necesidad de escribir código Java, esto es una conclusión natural.

Sin embargo SQL y Pig Latin tienen diferentes puntos fuertes y débiles. Debido a que Hive ofrece SQL, es una mejor herramienta para hacer análisis de datos tradicionales. La mayoría de los analistas de datos ya están familiarizados con SQL y herramientas de inteligencia de negocios esperan para hablar con las fuentes de datos en SQL. Pig Latin es una mejor opción en la construcción de un pipeline de datos o haciendo la investigación sobre los datos en bruto.

2.6. MapReduce

Framework para procesamiento de datos en paralelo.

Figura 29: Logo MapReduce. Ref. [31]

MapReduce es un nuevo modelo de programación diseñado para dar soporte a la computación paralela sobre grandes conjuntos de datos repartidos entre varios computadores. El nombre está inspirado en los nombres de dos importantes funciones de la programación funcional: *Map* y *Reduce*. MapReduce ha sido adoptado mundialmente como una implementación *open source* denominada Hadoop.

MapReduce se emplea en la resolución práctica de algunos algoritmos susceptibles de ser paralelizados. No obstante, MapReduce no es la solución a cualquier problema, igual que cualquier problema no puede ser resuelto eficientemente por MapReduce. Por regla general se abordan problemas con conjuntos de datos de gran tamaño, alcanzando los petabytes de tamaño. Es por esta razón por la que este framework suele ejecutarse en sistemas de archivos distribuidos.

El concepto MapReduce lo introdujo Google en 2004 en el paper "MapReduce: Simplified Data Processing on Large Clusters". Mapreduce fue usado por Hadoop, desarrollado inicialmente por Yahoo! y actualmente por Apache. Las primeras implementaciones de Google necesitaban realizar operaciones de multiplicación de grandes matrices para calcular el PageRank (ranking de páginas en una búsqueda). De esta forma se hizo popular MapReduce como un método de cálculo de álgebra lineal. La preocupación por tratar grandes colecciones de datos, llevó a crear algoritmos y frameworks capaces de poder procesar terabytes de información. Una de las primeras aplicaciones capaces de programar MapReduce fue Hadoop. A partir del año 2010 existen diversas iniciativas similares a Hadoop tanto en el ámbito de la industria como en el académico.

2.6.1. Características

Se han escrito implementaciones de bibliotecas de MapReduce en diversos lenguajes de programación como C++, Java y Python.

2.6.1.1. Funciones Map y Reduce

Las funciones Map y Reduce están definidas ambas con respecto a datos estructurados en tuplas del tipo (clave, valor).

Map()

Map toma uno de estos pares de datos con un tipo en un dominio de datos, y devuelve una lista de pares en un dominio diferente:

Map(k_1,v_1) -> list(k_2,v_2)

Se encarga del mapeo y es aplicada en paralelo para cada elemento de la entrada. Esto produce una lista de pares (k2,v2) por cada llamada. Después de ese primer paso junta todos los pares con la misma clave de todas las listas y los agrupa, creando un grupo para cada una de las diferentes claves generadas.

Reduce()

La función *Reduce* es aplicada en paralelo para cada grupo, produciendo una colección de valores para cada dominio:

Reduce(k_2, list (v_2)) -> list(v_3)

Cada llamada a *Reduce* produce un valor v3 o una llamada vacía, aunque una llamada puede retornar más de un valor. El retorno de todas esas llamadas se recoge como la lista de resultado deseado.

Por lo tanto, MapReduce transforma una lista de pares (clave, valor) en una lista de valores. Este comportamiento es diferente de la combinación "map and reduce" de programación funcional, que acepta una lista arbitraria de valores y devuelve un valor único que combina todos los valores devueltos por map.

2.6.1.2. Arquitectura

La función *map()* se ejecuta de forma distribuida a lo largo de varias máquinas. Los datos de entrada, procedentes por regla general de un gran archivo, se dividen en un conjunto de *M* particiones. Estas particiones pueden ser procesadas en diversas máquinas.

En una llamada a *MapReduce* ocurren varias operaciones:

- Se procede a dividir las entradas en M particiones de tamaño aproximado de 64MB. El programa MapReduce se comienza a instanciar en las diversas máquinas del cluster. Por regla general, el número de instancias se configura en las aplicaciones. Existen M tareas *map()* y R tareas *reduce()*.
- Una de las copias del programa toma el papel de "maestro". Al resto se les llama "workers", a estos el master les asigna tareas. Un worker tiene tres estados: reposo, trabajando, completo. El "maestro" se encarga de buscar "workers" en reposo (sin tarea asignada) y les asignará una tarea de map() o de reduce().

MAP

- Un worker con una tarea *map()* asignada tiene como entrada su partición correspondiente. Se dedicará a parsear los pares (clave, valor) para crear una nueva pareja de salida. Los pares clave y valor producidos por la función map() se almacenan como buffer en la memoria.

REDUCE

- Cada cierto tiempo los pares clave-valor almacenados en el buffer se escriben en el disco local, repartidos en R regiones. Las regiones de estos pares clave-valor son pasados al master, que es responsable de redirigir a los "workers" que tienen tareas de reduce().

- Cuando un worker de tipo reduce es notificado por el "maestro" con la localización de una partición, éste emplea llamadas remotas para hacer lecturas de la información almacenada en los discos duros por los workers de tipo map(). Cuando un worker de tipo reduce() lee todos los datos intermedios, ordena las claves de tal modo que se agrupan los datos encontrados que poseen la misma clave. El ordenamiento es necesario debido a que, por regla general, muchas claves de funciones map() diversas pueden ir a una misma función reduce(). En aquellos casos en los que la cantidad de datos intermedios sean muy grandes, se suele emplear un ordenamiento externo.

- El worker de tipo reduce() itera sobre el conjunto de valores ordenados intermedios, y lo hace por cada una de las claves únicas encontradas. Toma la clave y el conjunto de valores asociados a ella y se los pasa a la función reduce(). La salida de reduce() se añade al archivo de salida de MapReduce.

FINALIZACIÓN

- Cuando todas las tareas map() y reduce() se han completado, el "maestro" levanta al programa del usuario. Llegados a este punto la llamada MapReduce retorna el control al código de un usuario, en ese momento se consideran finalizadas las tareas.

- Las salidas se distribuyen en un fichero completo, o en su defecto se reparten en R ficheros. Estos R ficheros pueden ser la entrada de otro MapReduce o puede ser procesado por cualquier otro programa que necesite estos datos.

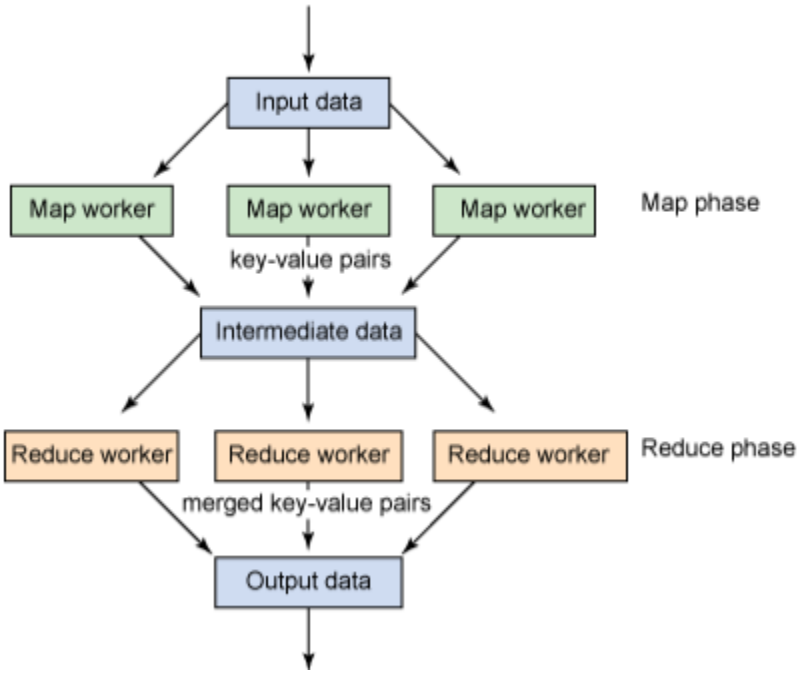


Figura 30: Flujo MapReduce simplificado.

La información escrita en local por los nodos workers de tipo map es agregada y ordenada por una función agregadora encargada de realizar esta operación. Los valores ordenados son de la forma [k, [v_1, v_2, v_3, ..., v_n]]. De esta forma la función reduce() recibe una lista de valores asociados a una única clave procedente del combinador. Debido a que la latencia de red de ordenadores y de sus discos suele ser mayor que cualquier otra de las operaciones, cualquier reducción en la cantidad de datos intermedios incrementará la eficiencia de los algoritmos, causando una mejora real de la eficiencia global. Es por esta razón que muchas distribuciones de MapReduce suelen incluir operaciones de agregación en local, mediante el uso de funciones capaces de agregar datos localmente. Evitando, o reduciendo en la medida de lo posible el movimiento de grandes ficheros.

La figura 34 refleja el comportamiento de MapReduce desde que el programa cliente crea la petición de trabajo hasta que se generan los archivos de salida:

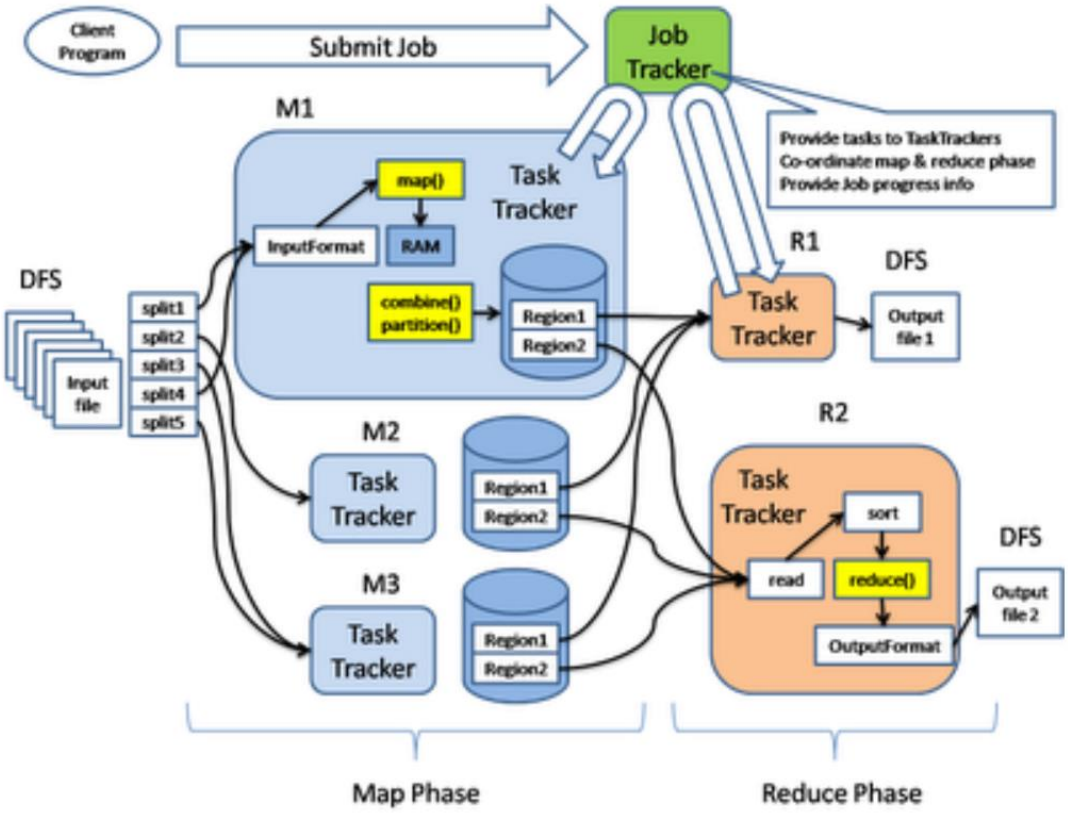


Figura 31: Flujo completo MapReduce. Ref. [32]

2.6.1.3. Tolerancia a fallos

Aun teniendo una baja probabilidad de fallo, es muy posible que uno (o varios) de los workers quede desactivo por ejemplo por fallo de la máquina que le daba soporte. El "master" periódicamente hace ping a cada worker para comprobar su estado. Si no existe respuesta tras un cierto tiempo de espera, el master interpreta que el worker está desactivado. Cualquier tarea map() completada por el worker vuelve de inmediato a su estado de espera, y por lo tanto puede resultar elegible para su asignación en otros workers. De forma similar, cualquier función map o reduce que se encuentre en progreso durante el fallo, se resetea a estado de reposo pudiendo ser elegida para su nueva re-asignación.

Las tareas map() completadas se vuelven a re-ejecutar si hay un fallo debido en parte a que su salida se almacena en los discos locales de la máquina que falló, y por lo tanto se consideran inaccesibles. Las tareas reduce() completas no es necesario que vuelvan a ser ejecutadas debido a que su salida se ha almacenado en el sistema global. Cuando la tarea de map() es ejecutada por un *worker A* y luego por un worker B (debido principalmente a un fallo), en este caso todas las tareas reduce() son notificadas para que eliminen datos procedentes del *worker A* y acepten las del worker B.

2.6.2. Ejemplo MapReduce: Conteo de palabras

Función Map(): Va contando cada palabra que hay en *document* y poniendo un valor 1 asociado a la palabra, el cual simboliza una aparición de esa palabra en el documento.

```
map(String name, String document):
        // clave: nombre del documento
        // valor: contenido del documento
        for each word w in document:
                EmitIntermediate(w, 1);
```

Función Reduce(): Para buscar las apariciones de la palabra en *cuentasParciales* y sumar todos los valores asociados en la variable *result*.

```
reduce(String word, Iterator cuentasParciales):
        // word: una palabra
        // cuentasParciales: una lista parcial para cuentas agregadas
        int result = 0;
        for each v in cuentasParciales:
                result += ParseInt(v);
        Emit(result);
```

Ejemplo: MAP

Nos llega a las funciones map dos particiones de un archivo:

- (f1, “El perro estaba en el parque ladrando”)
- (f2, “El niño de Juan estaba llorando esta noche en la cuna”)

Lo que produciría la siguiente salida de las funciones map:

- (‘el’, 1), (‘perro’, 1), (‘estaba’, 1), (‘en’, 1), (‘el’, 1), (‘parque’, 1), (‘ladrando’, 1), (‘el’, 1), (‘niño’, 1), (‘de’, 1), (‘Juan’, 1), (‘estaba’, 1), (‘llorando’, 1), (‘esta’, 1), (‘noche’, 1), (‘en’, 1), (‘la’, 1), (‘cuna’, 1).

FUNCIÓN INTERMEDIA

Produce lo siguiente juntando los pares de misma clave (palabra):

- (‘el’, [1, 1, 1]), (‘perro’, [1]), (‘estaba’, [1, 1]), (‘en’, [1, 1]), (‘parque’, [1]), (‘ladrando’, [1]), (‘niño’, [1]), (‘de’, [1]), (‘Juan’, [1]), (‘llorando’, [1]), (‘esta’, [1]), (‘noche’, [1]), (‘la’, [1]), (‘cuna’, [1]).

REDUCE

Realizará el conteo para cada par asociado a una palabra:

- (‘el’, 3), (‘perro’, 1), (‘estaba’, 2), (‘en’, 2), (‘parque’, 1), (‘ladrando’, 1), (‘niño’, 1), (‘de’, 1), (‘Juan’, 1), (‘llorando’, 1), (‘esta’, 1), (‘noche’, 1), (‘la’, 1), (‘cuna’, 1).

3. Objetivos

El objetivo principal del software easyMahout es ofrecer al usuario una manera sencilla e intuitiva de ejecutar algoritmos de Machine Learning sobre el framework Apache Hadoop. Partimos de la base de que el usuario tiene pocas nociones teóricas de algoritmos de Machine Learning como son los sistemas de recomendación, algoritmos de clustering o algoritmos de clasificación, aunque un usuario experimentado en el tema también saldría beneficiado del uso de nuestra aplicación ahorrando tiempo en la configuración y ejecución de estos algoritmos. Por esta razón, la aplicación debe ser simple y clara. No es nuestra intención ofrecer al usuario un entorno de desarrollo tan completo como podría ser Eclipse, Adobe Photoshop o el propio MS Office.

La clave de nuestra aplicación es el uso del framework Apache Mahout (biblioteca de algoritmos inteligentes programados bajo el modelo de MapReduce) en conjunción con Apache Hadoop (permite la ejecución de programas implementados en MapReduce sobre sistema altamente distribuidos y en millares de nodos). Hasta ahora, la instalación, configuración y uso de Mahout y Hadoop se presentaba prácticamente imposible para un usuario con bajos conocimientos de GNU/LINUX, principalmente por el uso imprescindible de la SHELL para hacer ejecutar dichos algoritmos.

Resumiendo, los usuarios con un nivel bajo, medio y alto en las aptitudes arriba descritas, podrán utilizar Mahout de una manera fácil y eficaz, omitiendo pasos que de otra manera hubieran sido necesarios como leer interminables libros especializados en la materia.

Claves de easyMahout:

- Algoritmos de clustering
    - Tener disponible en nuestro entorno de ejecución la mayoría de los algoritmos de clustering presentes en Mahout.
    - Ofrecer la posibilidad de ejecutar tareas de clustering en modo distribuido (mediante Apache Hadoop).
- Sistemas de recomendación
    - Ofrecer la posibilidad de ejecutar los algoritmos de recomendación implementados en Apache Mahout.
    - Poder ejecutar tareas de recomendación en modo distribuido (mediante Apache Hadoop)
- Algorítmos de clasificación
    - Ofrecer la posibilidad de ejecutar algoritmos de clasificación disponibles en Apache Mahout.
    - Poder ejecutar tareas de clasificacion en modo distribuido (mediante Apache Hadoop)

4. Arquitectura propuesta

4.1. Arquitectura del sistema

La aplicación *easyMahout* está construida sobre el framework Apache Mahout. Principalmente es una nueva capa más externa en la arquitectura, que brinda a Mahout y sus capas internas de una interfaz gráfica de usuario con la que poder desarrollar las funciones propias de Mahout. A su vez, y de manera sucesiva, Mahout es una capa de funcionalidad construida sobre Apache Hadoop, el cual trabaja con MapReduce y sobre el sistema de ficheros HDFS. El diagrama global de la arquitectura de *easyMahout* es el siguiente:

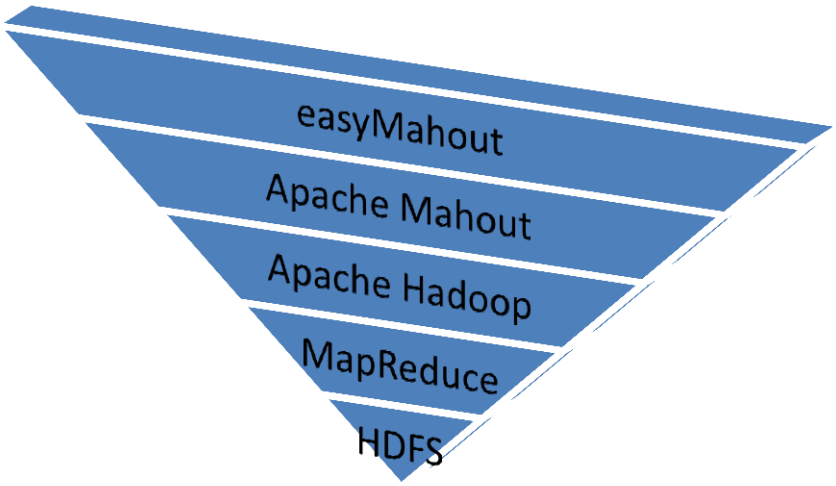


Figura 32: Arquitectura de easyMahout.

Podemos dividir easyMahout en tres partes bien diferenciadas según la funcionalidad de Mahout que cubren, estas tres secciones son: Sistemas de recomendación, clasificación y clustering. Cada una de estas secciones trabaja con los algortimos de mahout correspondientes. Además de lo anterior, cabe explicar que la arquitectura de Mahout, a su vez, está dividida en dos: distribuida y no distribuida o local. Cada algoritmo está organizado según el tipo de implementación llevado a cabo. Si este esta implementado mediante MapReduce, será distribuido, mientras que la implementación básica será destinada a un uso del algoritmo local.

La estructura detallada de cada sección de easyMahout se muestra gracias a los siguientes diagramas.

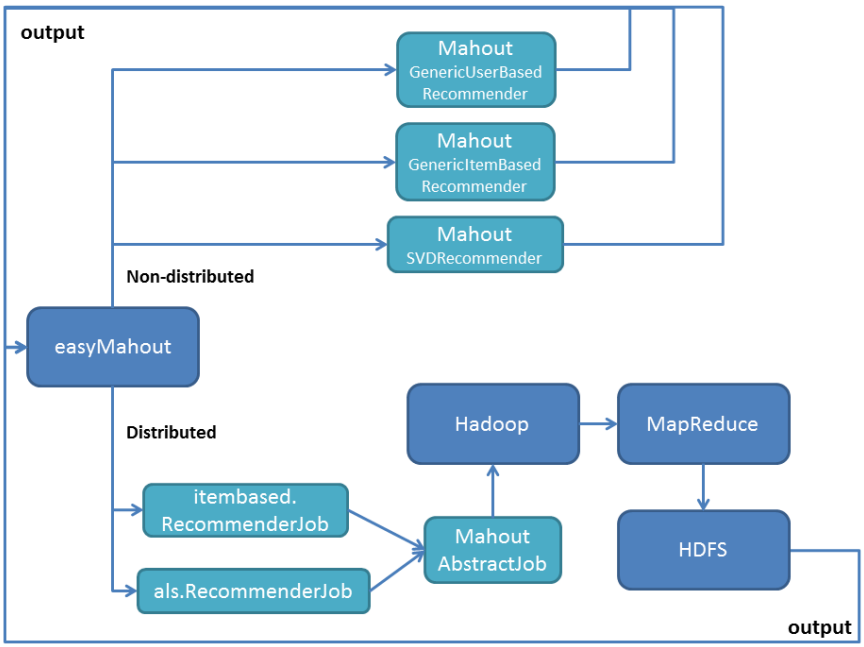


Figura 33: Arquitectura de los sistemas de recomendación de easyMahout.

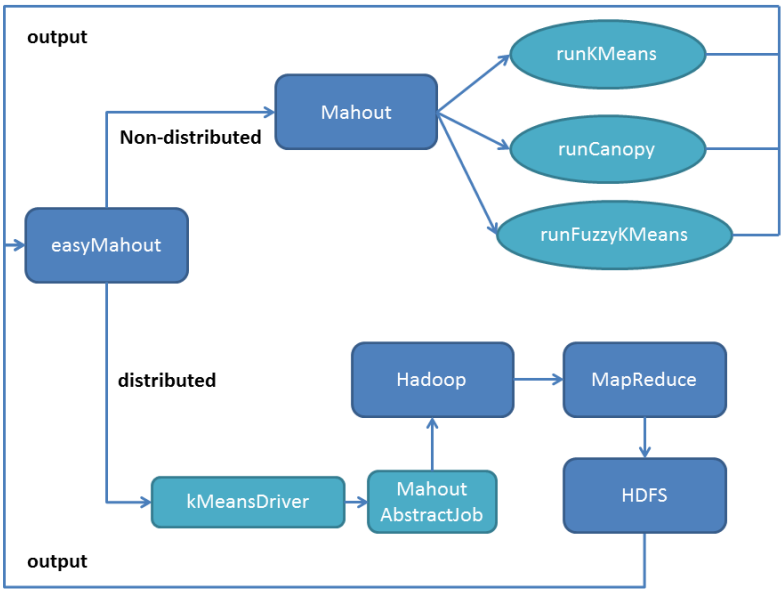


Figura 34: Arquitectura del clustering de easyMahout.

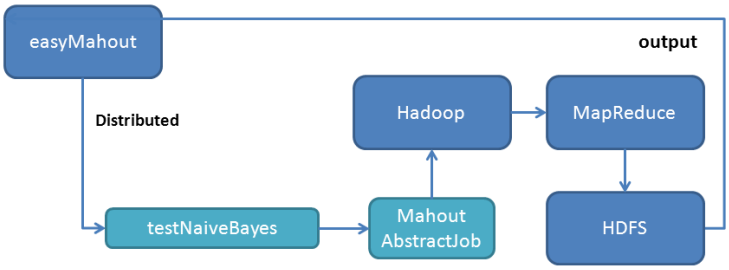


Figura 35: Arquitectura de los clasificadores de easyMahout.

4.2. Casos de uso

4.2.1. Ejecutar ALS-WR recommender

Actor principal: Usuario.

Objetivo en contexto: El usuario ejecuta el algoritmo de recomendación ALS-WR. **Precondiciones:** El usuario tiene su sistema correctamente configurado, así como las variables de entorno para la ejecución y selecciona la pestaña *Recommender* dentro de la interfaz.

Disparador: El usuario decide ejecutar el algoritmo de recomendación por factorización de matrices ALS-WR seleccionándolo desde la interfaz.

Escenario:

1- El usuario ejecuta el algoritmo ALS-WR desde la aplicación.

Excepciones:

1- Usuario no introduce los datos adecuados en los campos que tiene que rellenar.

Prioridad: Alta.

Frecuencia de uso: Alta.

Canal para el actor: A través de la interfaz de la aplicación.

Flujo de eventos:

Flujo básico:

1- El usuario ejecuta la aplicación y visualiza la interfaz de la misma.

1.1-**Subflujo ejecutarUserBasedRecommender:**

Permite al usuario ejecutar el algoritmo de recomendación basada en usuario sobre sus datos de entrada.

1.2-**Subflujo ejecutarItemBasedRecommender**:

Permite al usuario ejecutar el algoritmo de recomendación basada en objeto sobre sus datos de entrada.

1.3-**Subflujo ejecutarFactorizedRecommender**:

Permite al usuario ejecutar el algoritmo de recomendación por factorización de matrices sobre sus datos de entrada.

Novela

"El **usuario** abre la **aplicación,** se sitúa en la pestaña *Recommender* y selecciona el **algoritmo** *Factorization Matrix Recommender* del nodo "Type" del árbol correspondiente de la interfaz. A continuación rellena los campos de los paneles modelo de datos, similitud, factorización, y rellena las predicciones deseadas en el panel consultas desde donde ejecuta el **algoritmo**. El **sistema** revisará tras cada paso si los **datos** introducidos son correctos y en caso de que no, le mostrará el correspondiente **mensaje de error**."

Clases potenciales*:	**Clasificación general:**
Usuario	Entidad externa.
Sistema	Entidad.
Tipo de sistema de recomendación	Entidad externa.
Modelo de datos	Entidad externa.
Métrica de similitud	Entidad externa.
Factorización	Entidad externa.
Consultas	Entidad externa.

* Se considera: aplicación = sistema.

Para que se considere una clase legítima para su inclusión en el modelo de requerimientos debe satisfacer una serie de características.

Clases potenciales*:	**Aceptada/rechazada:**
Usuario	Rechazada.
Sistema	Rechazada.
Tipo de sistema de recomendación	Aceptada.
Modelo de datos	Aceptada.
Métrica de similitud	Aceptada.
Factorización	Aceptada.
Consultas	Aceptada.

Diagrama de flujo:

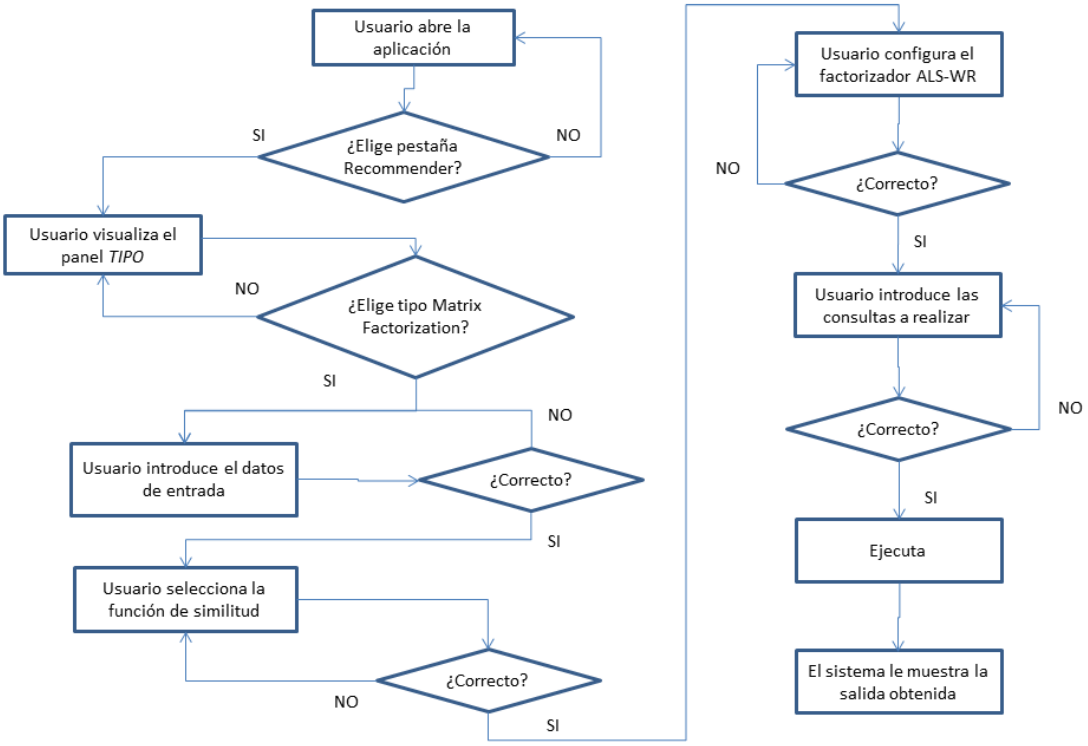


Figura 36: Diagrama de flujo del sistema de recomendación por factorización de matrices.

4.2.2. Ejecutar K-Means clustering

Actor principal: Usuario.

Objetivo en contexto: El usuario ejecuta el algoritmo de clustering K-Means.
Precondiciones: El usuario tiene su sistema correctamente configurado y selecciona la pestaña Clustering dentro de la interfaz.

Disparador: El usuario decide ejecutar el algoritmo K-Means seleccionándolo desde la interfaz.
Escenario:

1- El usuario ejecuta el algoritmo K-Means en la aplicación.

Excepciones:

1- Usuario no introduce los datos adecuados en los campos que tiene que rellenar.

Prioridad: Alta.

Frecuencia de uso: Alta.

Canal para el actor: A través de la interfaz de la aplicación.

Flujo de eventos:

Flujo básico:

1- El usuario ejecuta la aplicación y visualiza la interfaz de la misma

1.1-**Subflujo ejecutarKMeans:**

Permite al usuario ejecutar el algoritmo K-Means sobre sus datos de entrada.

1.2-**Subflujo ejecutarCanopy**:

Permite al usuario ejecutar el algoritmo Canopy sobre sus datos de entrada.

1.3-**Subflujo ejecutarFuzzyKMeans**:

Permite al usuario ejecutar el algoritmo Fuzzy K-Means sobre sus datos de entrada.

Novela

"El **usuario** abre la **aplicación** y selecciona el **algoritmo** K-Means del nodo "Algorithm" del árbol correspondiente de la interfaz. A continuación rellena los campos de los paneles de medida de distancia de similitud entre objetos, umbral de convergencia, numero de clústeres que desea formar la salida, máximo de iteraciones que el algoritmo tiene que dar y modelo de datos desde donde ejecuta la **aplicación**. El **sistema** revisará tras cada paso si los **datos** introducidos son correctos y en caso de que no, le mostrará el correspondiente **mensaje de error**."

Clases potenciales*:	**Clasificación general:**
Usuario	Entidad externa.
Sistema	Entidad.
Algoritmo	Entidad externa.
Medida de distancia	Entidad externa.
Umbral de convergencia	Entidad externa.
Numero de clústeres	Entidad externa.
Máximo iteraciones	Entidad externa.
Modelo de datos	Entidad externa.

* Se considera: aplicación = sistema.

Para que se considere una clase legítima para su inclusión en el modelo de requerimientos debe satisfacer una serie de características.

Clases potenciales*:	**Aceptada/rechazada:**
Usuario	Rechazada.
Sistema	Rechazada.
Algoritmo	Aceptada.
Medida de distancia	Aceptada.
Umbral de convergencia	Aceptada.
Numero de clústeres	Aceptada.
Máximo iteraciones	Aceptada.
Modelo de datos	Aceptada.

Diagrama de flujo:

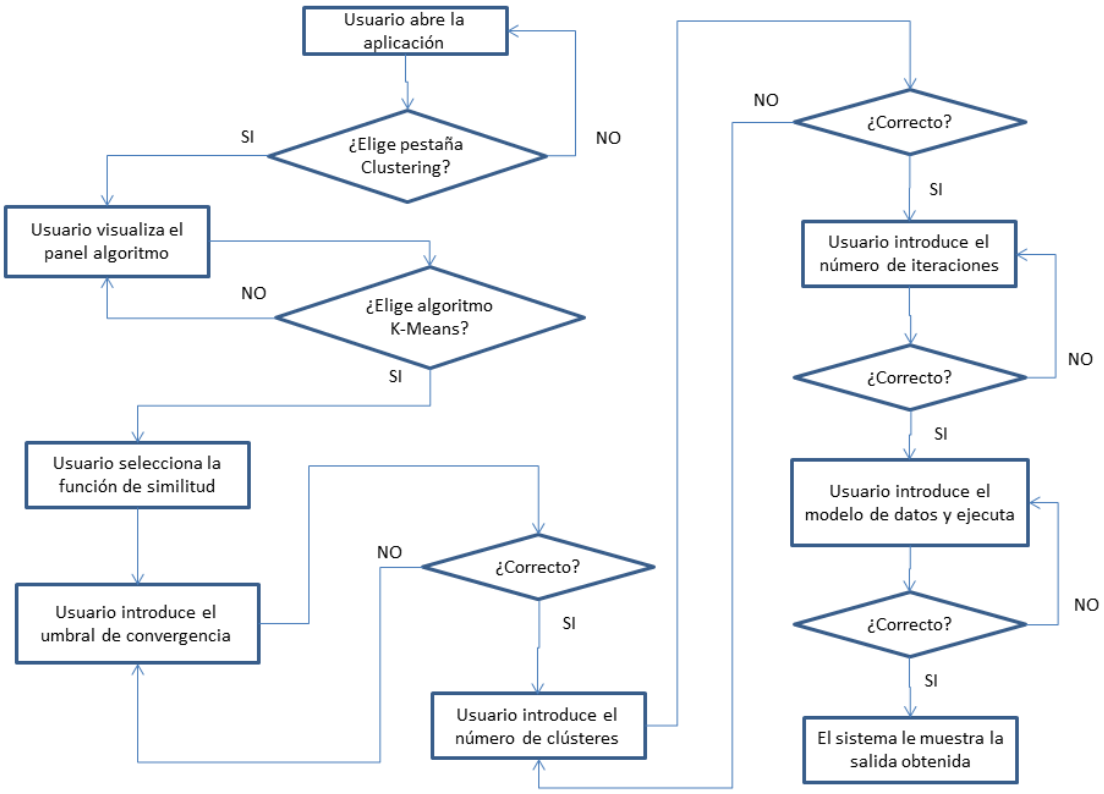


Figura 37: Diagrama de flujo del algoritmo de clustering K-Means.

4.2.3. Ejecutar Naive Bayes

Actor principal: Usuario.

Objetivo en contexto: El usuario ejecuta el algoritmo de clasificación *Naive Bayes*.
Precondiciones: El usuario tiene su sistema correctamente configurado, así como las variables de entorno para la ejecución.

Disparador: El usuario selecciona la pestaña *Classification* dentro de la interfaz.

Escenario:

1- El usuario ejecuta el algoritmo *Naive Bayes* en la aplicación.

Excepciones:

1- Usuario no introduce los datos adecuados u obligatorios en los campos que tiene que rellenar.

Prioridad: Alta.

Frecuencia de uso: Alta.

Canal para el actor: A través de la interfaz de la aplicación.

Flujo de eventos:

Flujo básico:

1- El usuario ejecuta la aplicación y visualiza la interfaz de la misma

1.1-**Subflujo ejecutar Naive Bayes:**

Permite al usuario ejecutar el algoritmo Naive Bayes sobre sus datos de entrada.

1.2-**Subflujo ejecutar Complementary Naive Bayes**:

Permite al usuario ejecutar el algoritmo Complementary Naive Bayes sobre sus datos de entrada.

Novela

"El **usuario** abre la **aplicación**, elige la pestaña Classification, después selecciona el **algoritmo** Naive Bayes del nodo "Algorithm" del árbol correspondiente de la interfaz. A continuación rellena los campos de los paneles de selección de datos a usar como test y el modelo de datos desde donde ejecuta la **aplicación** pulsando el botón Run Classifier. El **sistema** revisará tras cada paso si los **datos** introducidos son correctos y en caso de que no, le mostrará el correspondiente **mensaje de error**."

Clases potenciales*:	**Clasificación general:**
Usuario	Entidad externa.
Sistema	Entidad.
Tipo de sistema de clasificación	Entidad externa.
Datos de Test	Entidad externa.
Modelo de datos	Entidad externa.
Consultas	Entidad externa.

* Se considera: aplicación = sistema.

Para que se considere una clase legítima para su inclusión en el modelo de requerimientos debe satisfacer una serie de características.

Clases potenciales*:	**Aceptada/rechazada:**
Usuario	Rechazada.
Sistema	Rechazada.
Tipo de sistema de clasificación	Aceptada.
Datos de test	Aceptada.
Modelo de datos	Aceptada.
Consultas	Aceptada.

Diagrama de flujo:

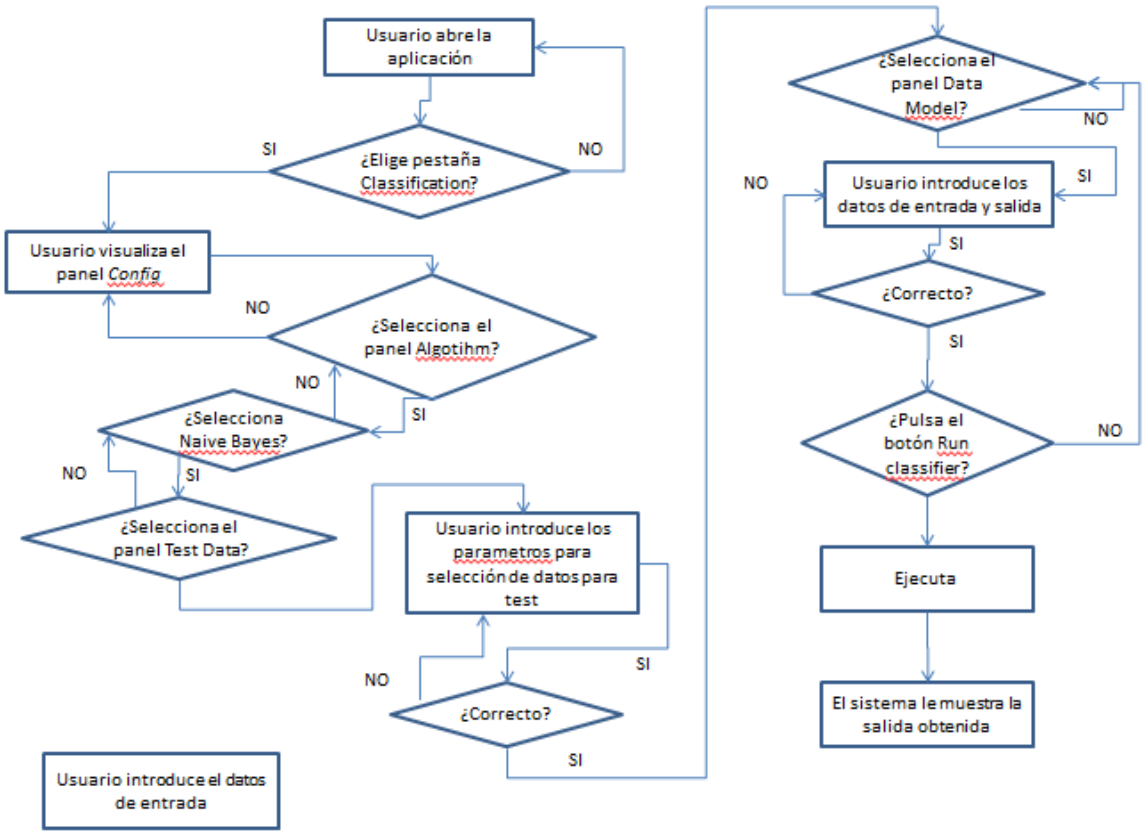
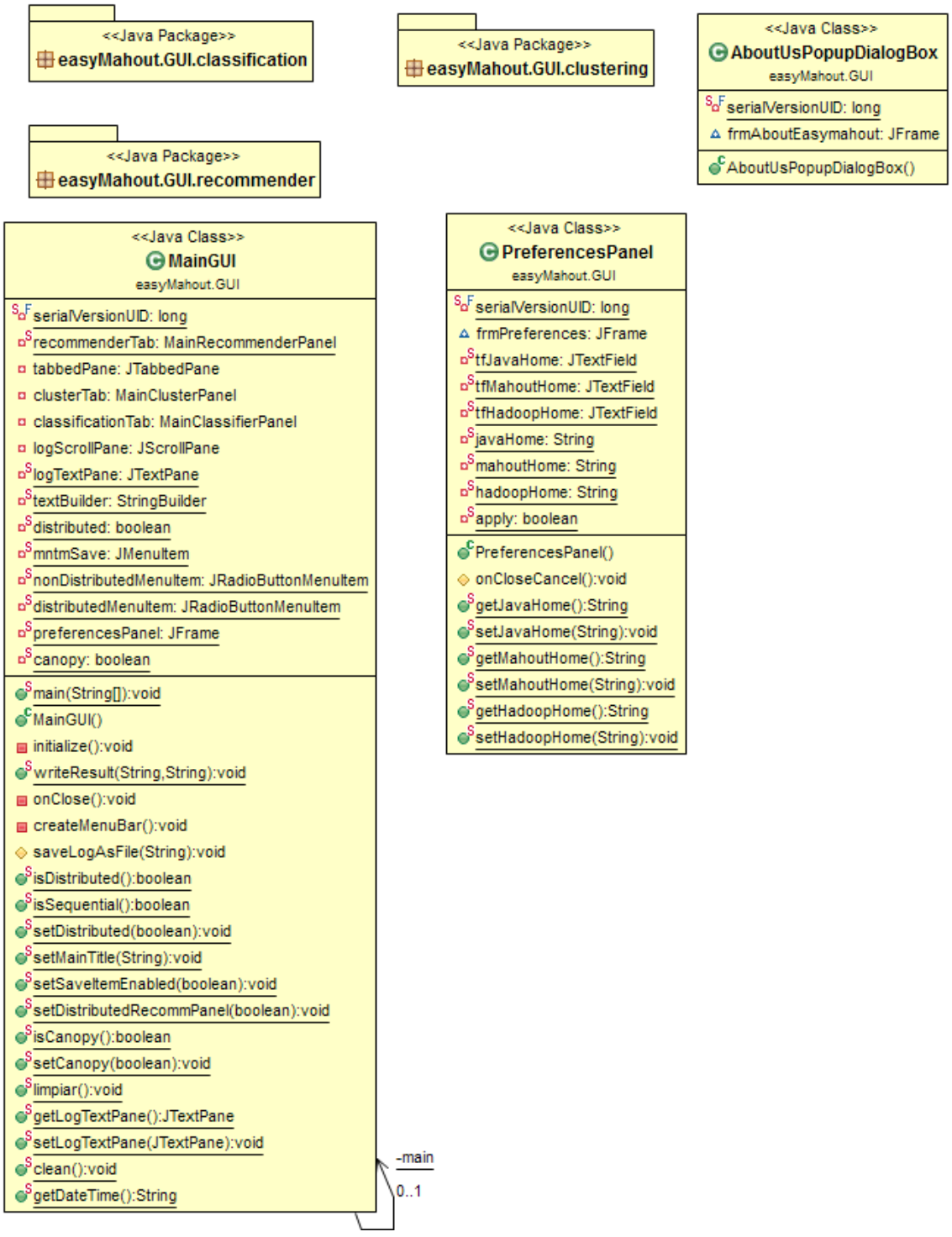


Figura 38: Diagrama de flujo de algoritmo de clasificación Naive Bayes.

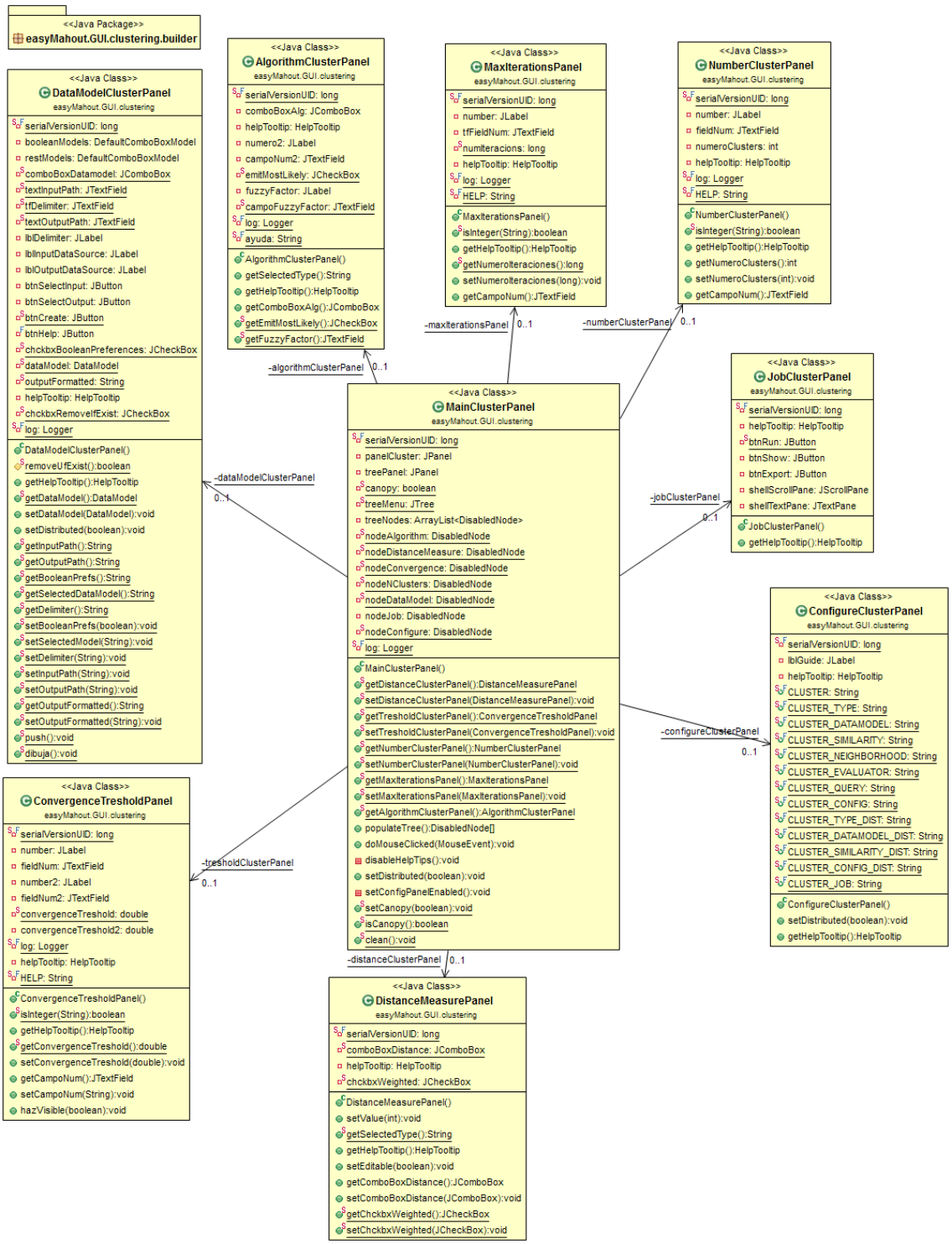
4.3. Diagramas de clases

A continuación presentamos la estructura del proyecto mediante diagramas de clases UML estructurados por paquetes. Existen 2 paquetes principales: GUI y Utils.

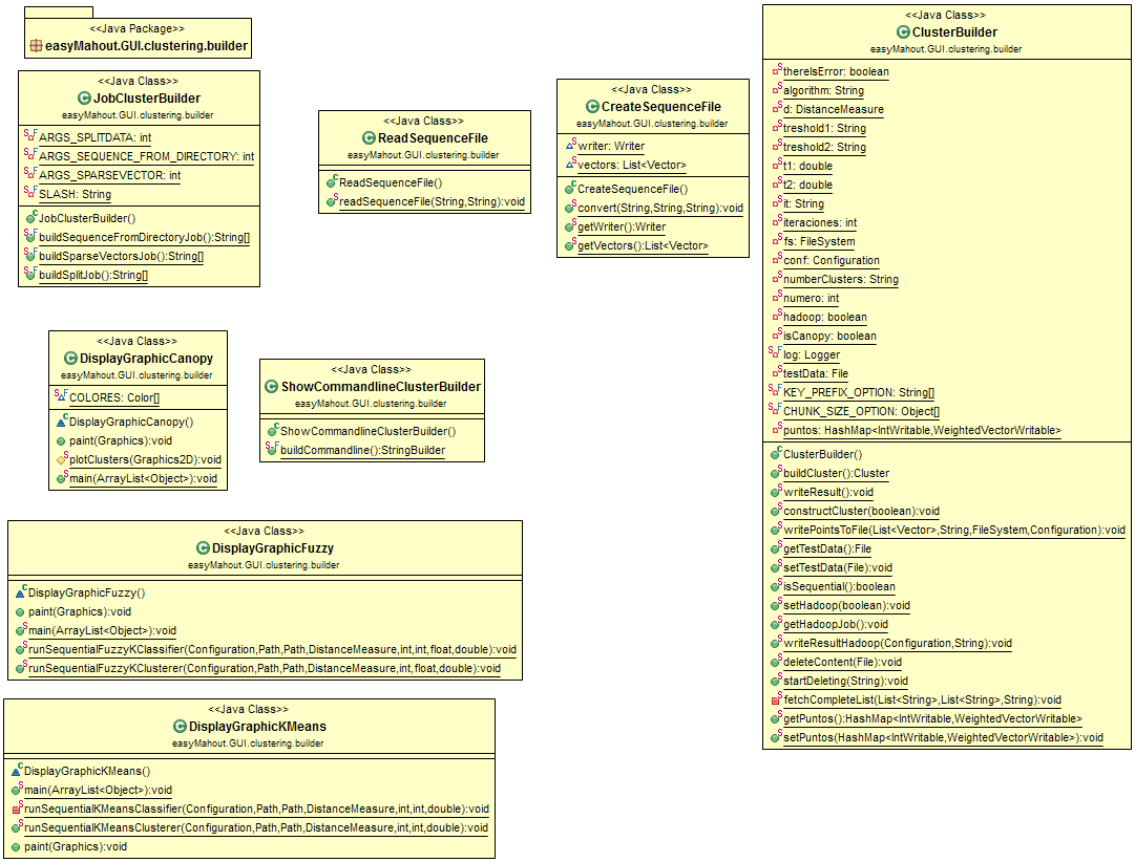
GUI



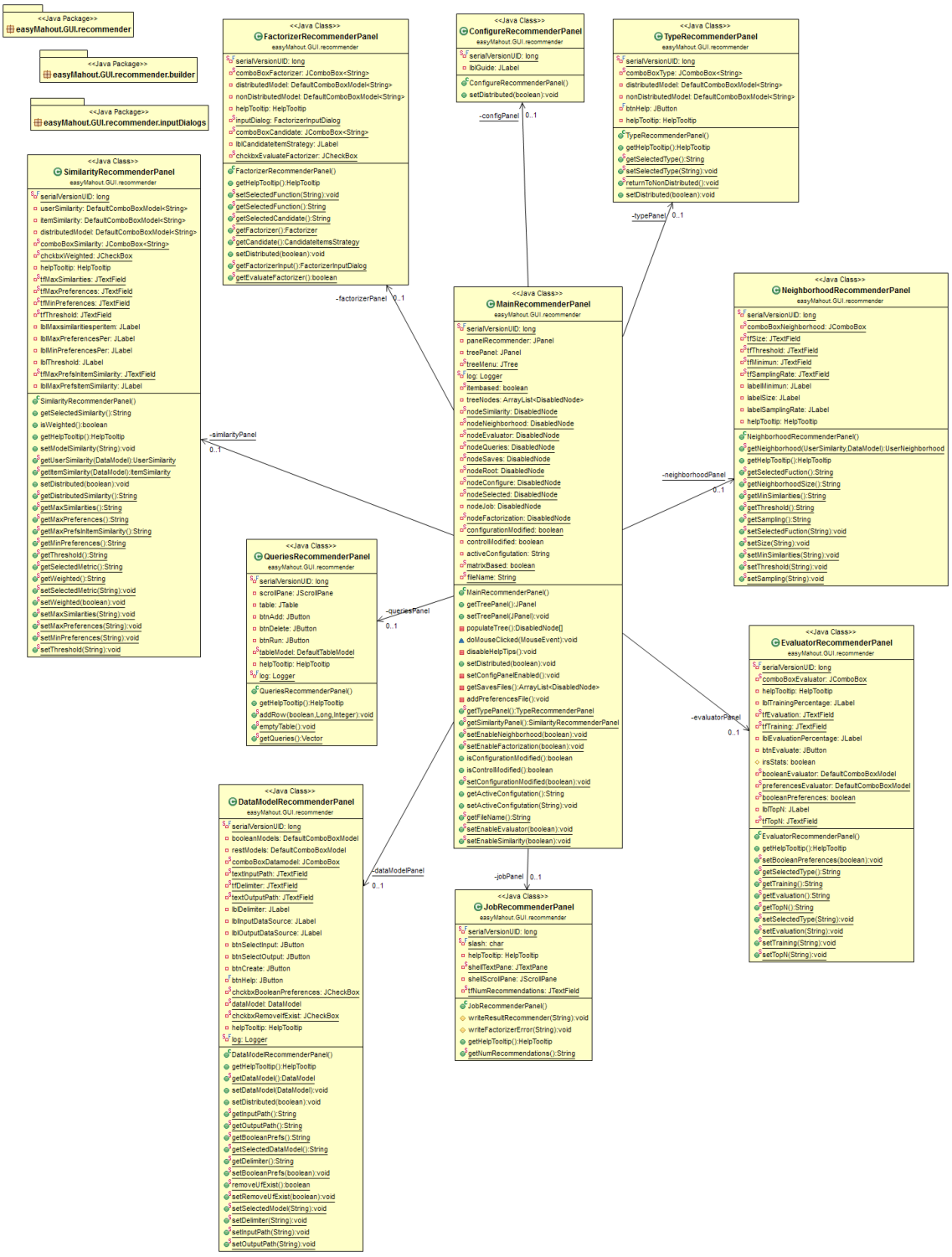
GUI.Clustering:



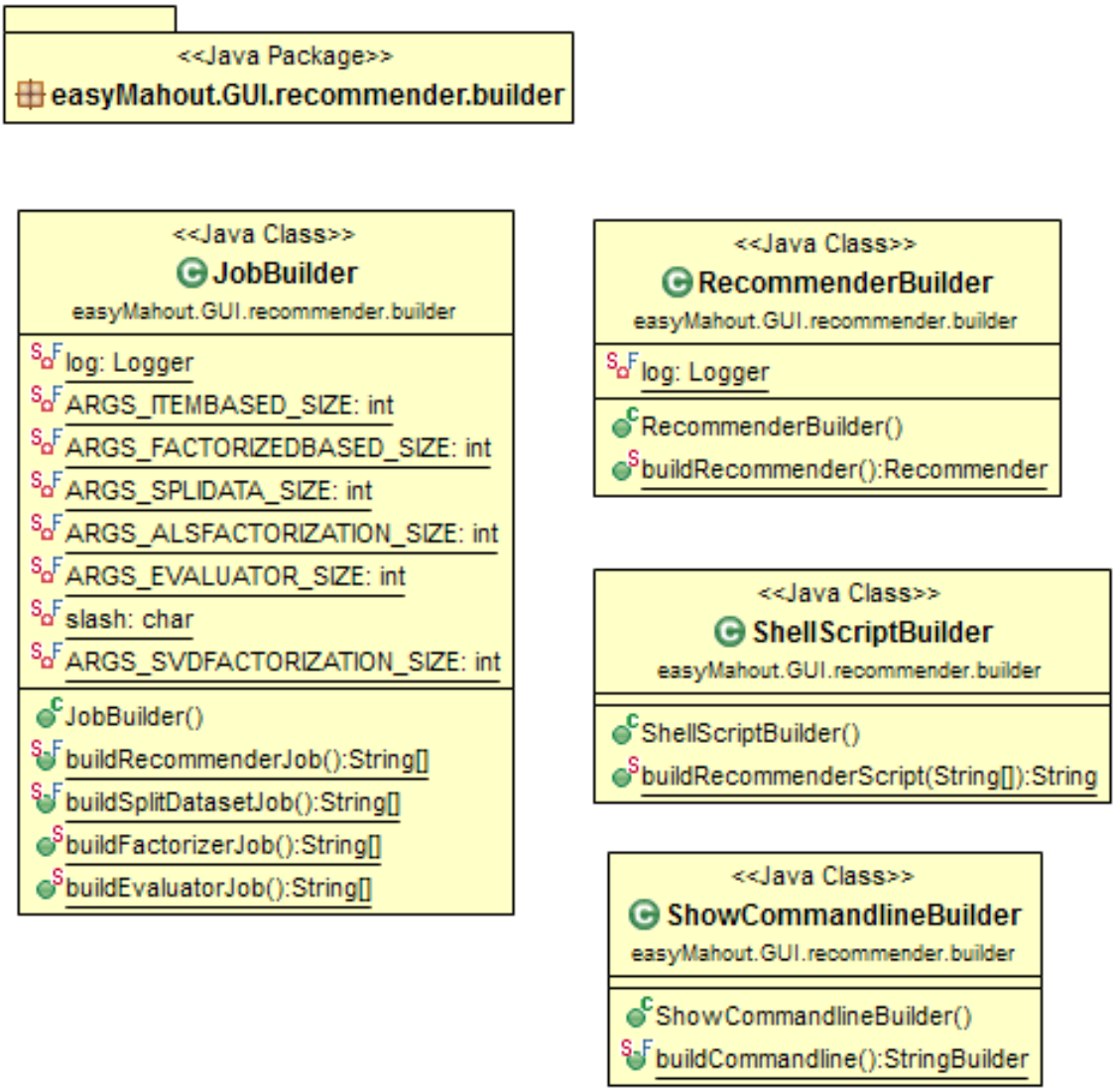
GUI.Clustering.builder: paquete usado para construir las tareas Mahout o Hadoop.



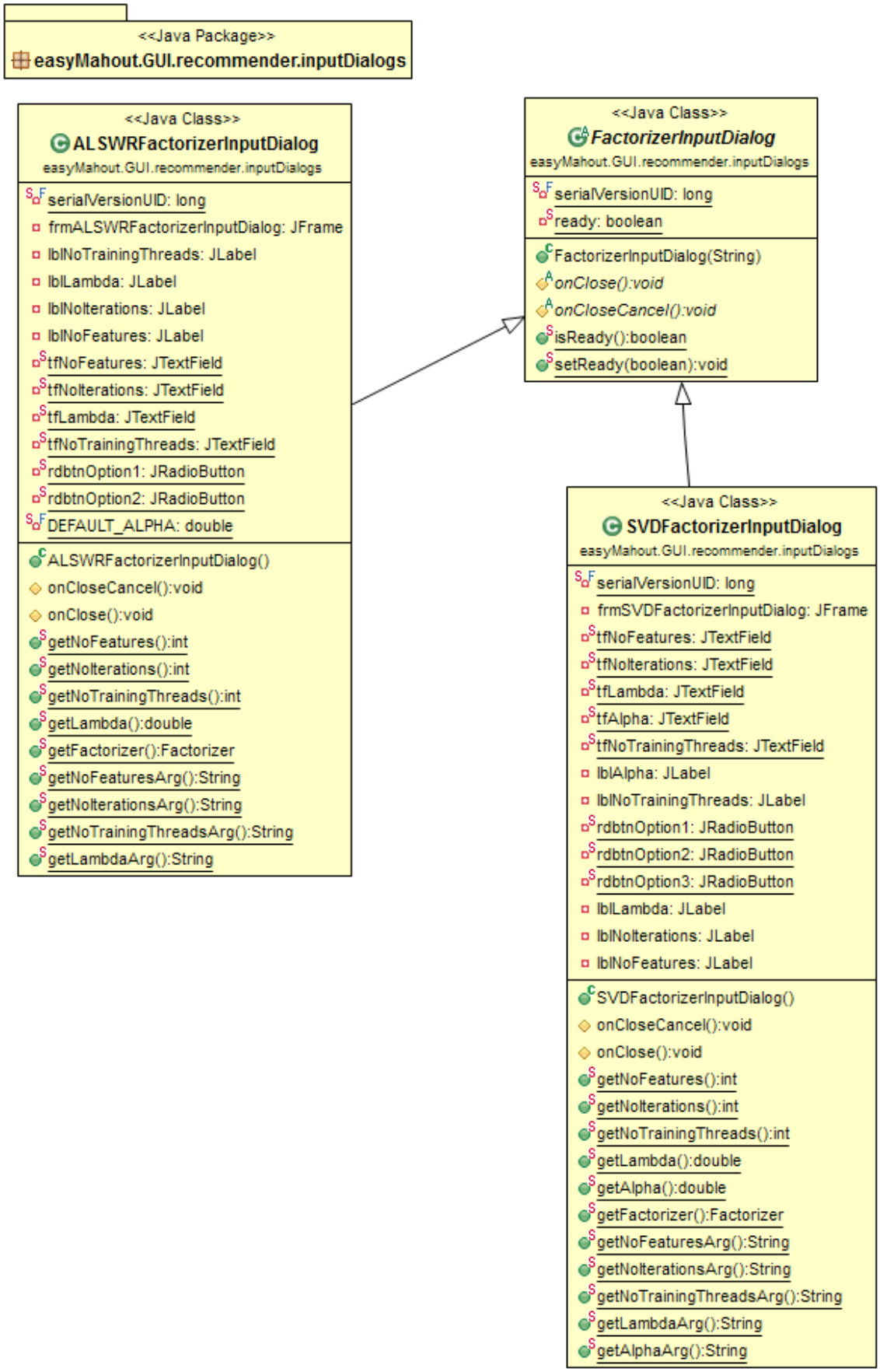
GUI.Recommender:



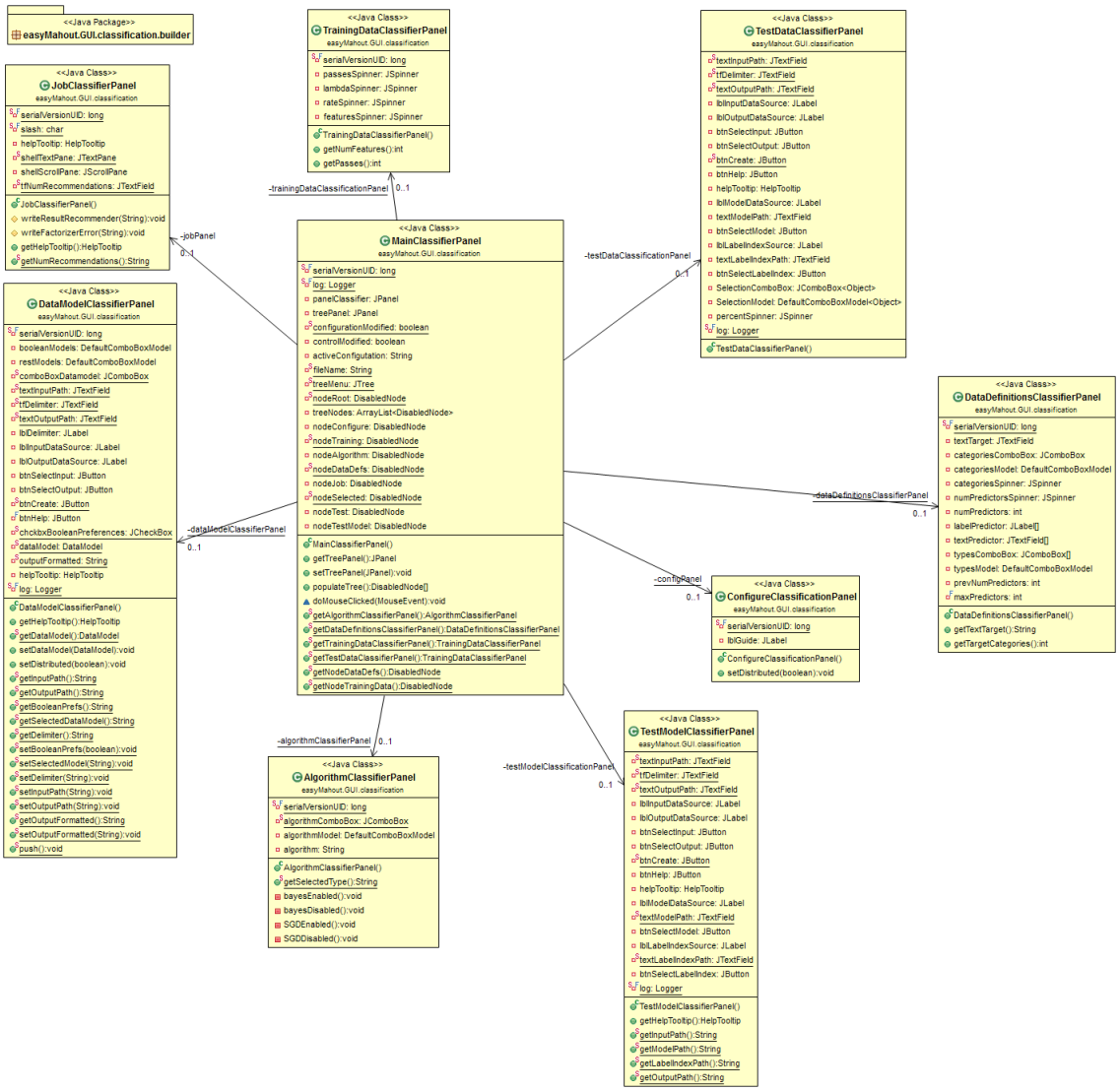
GUI.Recommender.builder:



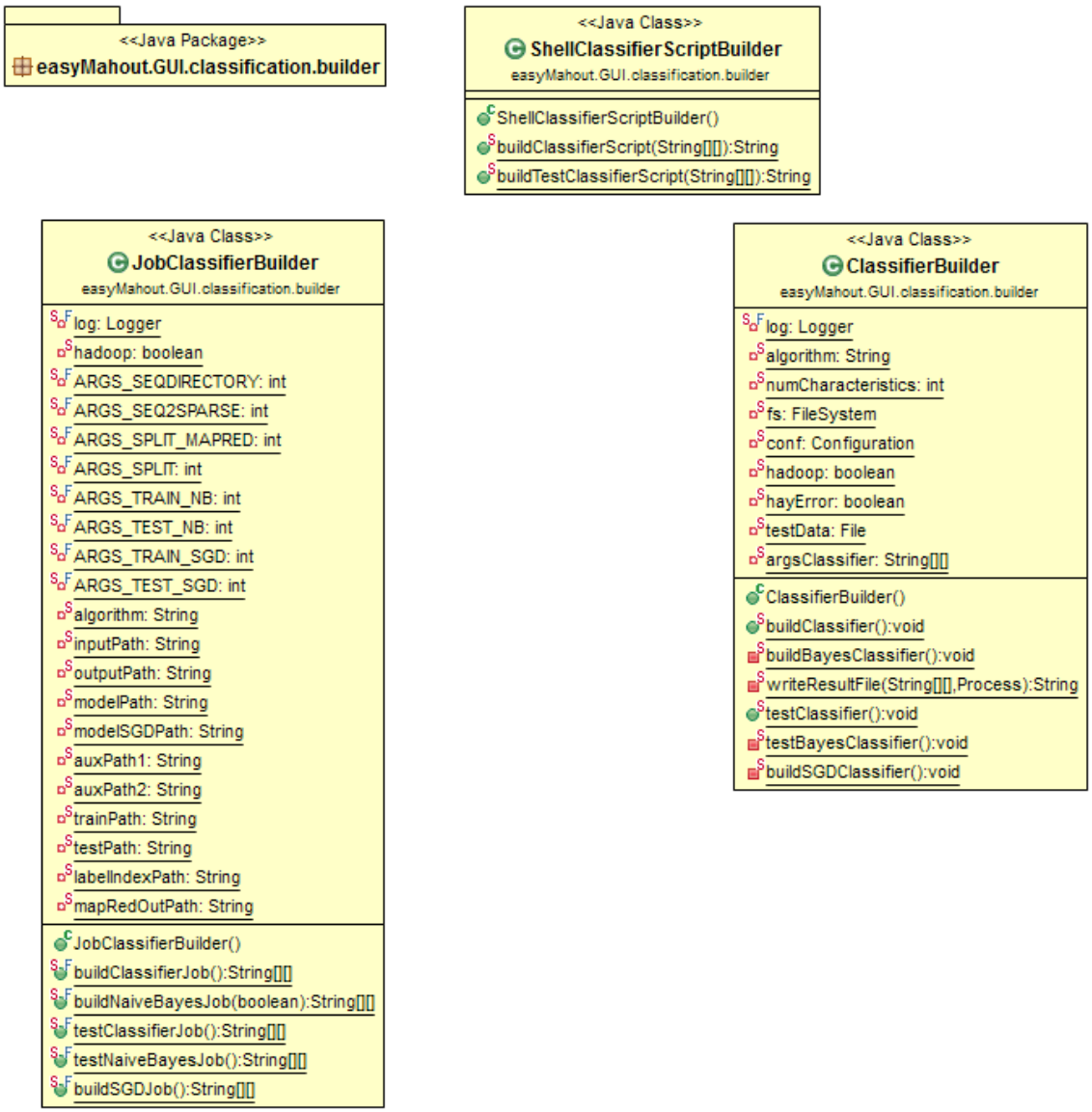
GUI.Recommender.InputDialogs:



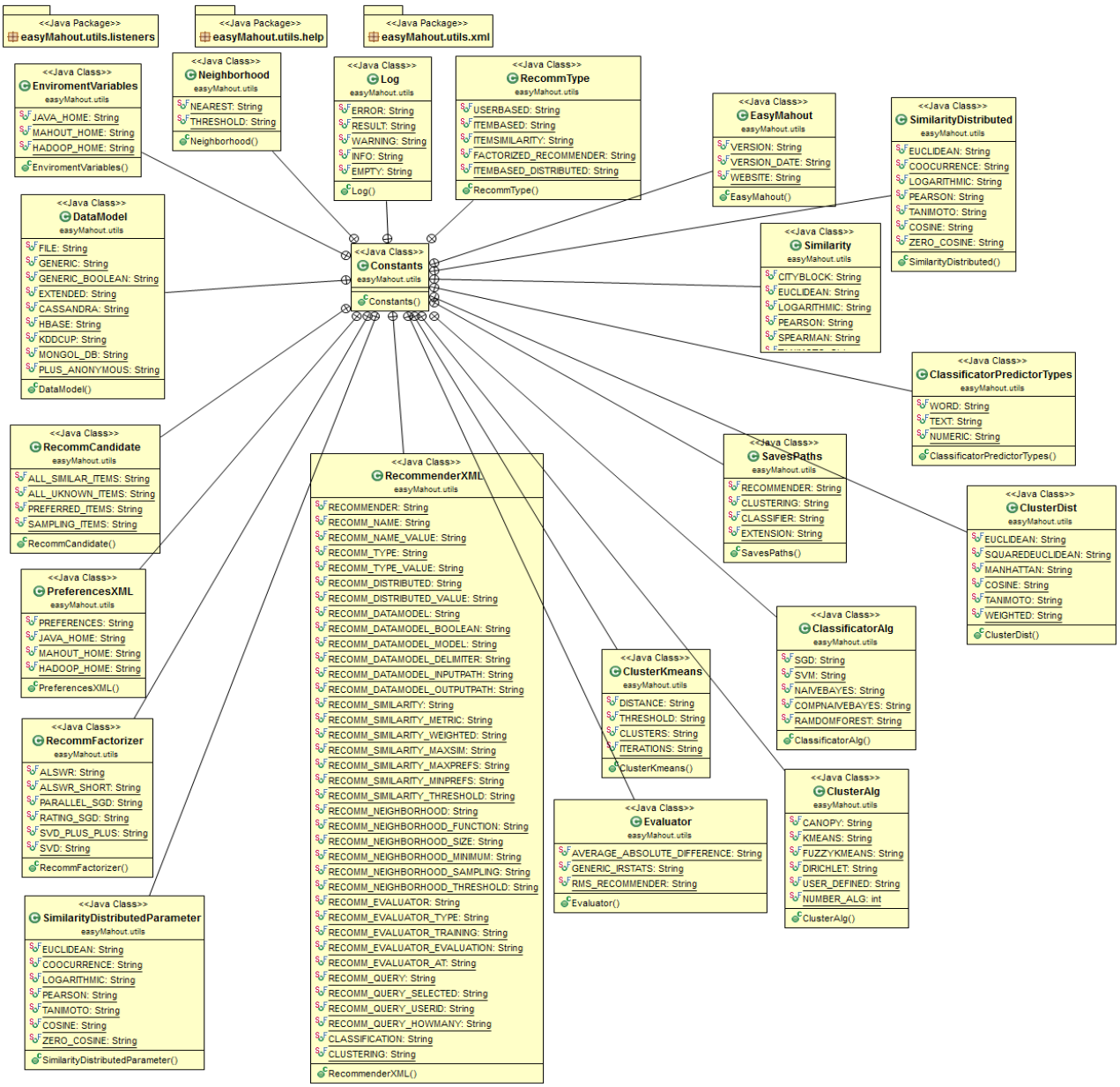
GUI.classification:



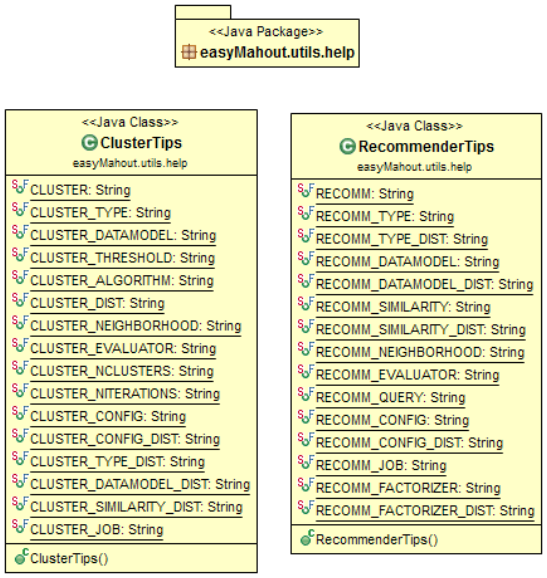
GUI.classification.builder:



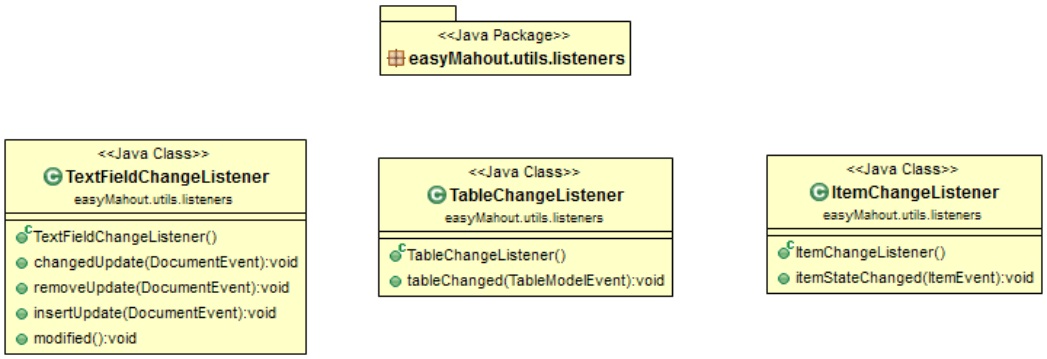
Utils es el otro paquete principal de la estructura del proyecto:



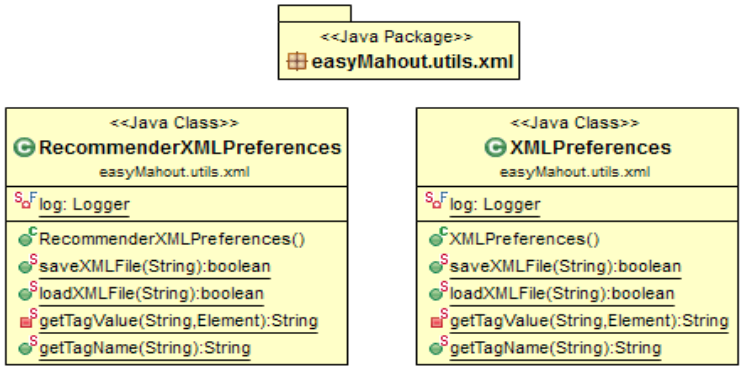
Utils.help:



Utils.Listeners:



Utils.xml:



4.4. Diagramas de secuencia

Algoritmo de recomendación ALS-WR

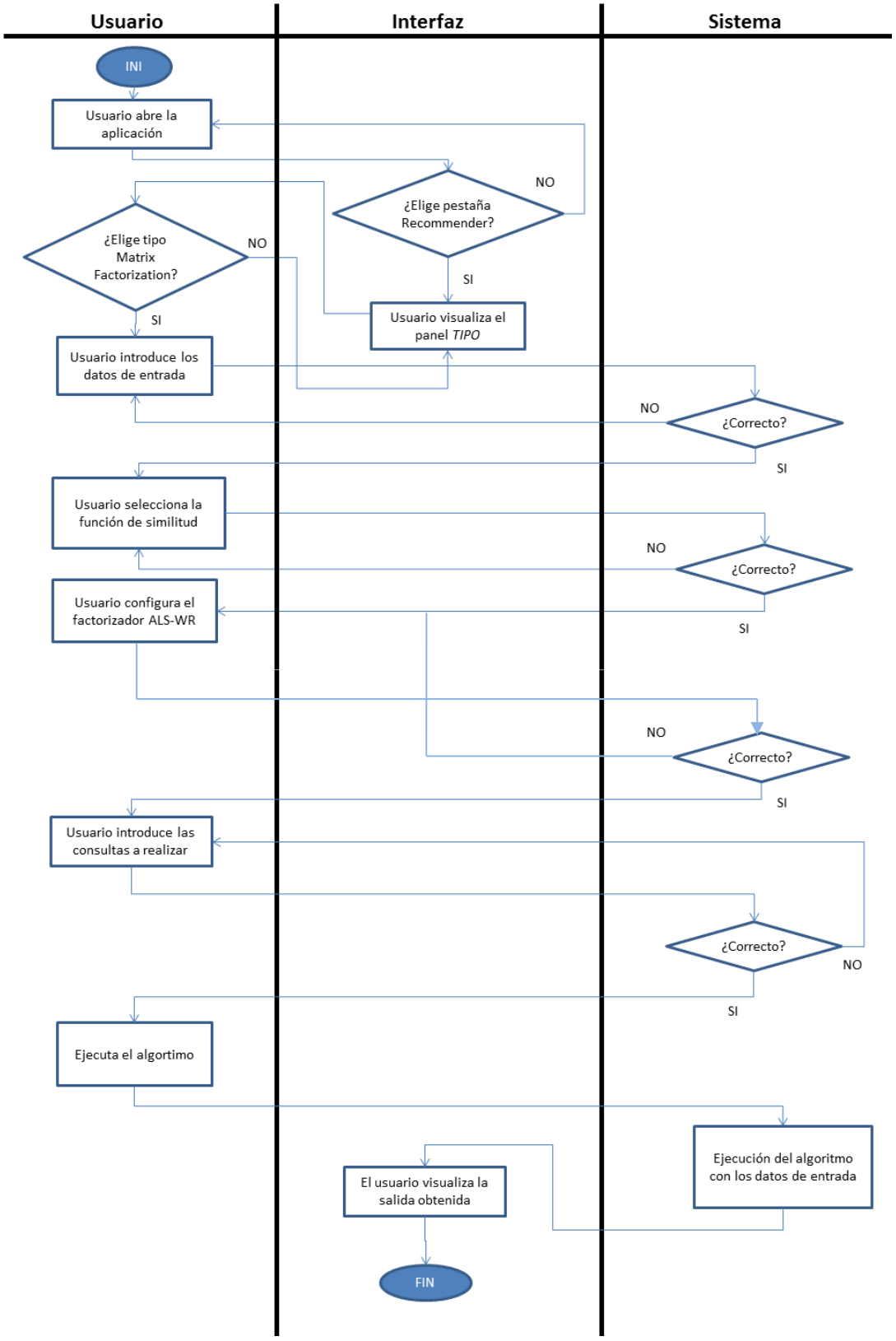


Figura 39: Diagrama de secuencia del sistema de recomendación por factorización de matrices.

Algoritmo de clustering K-Means

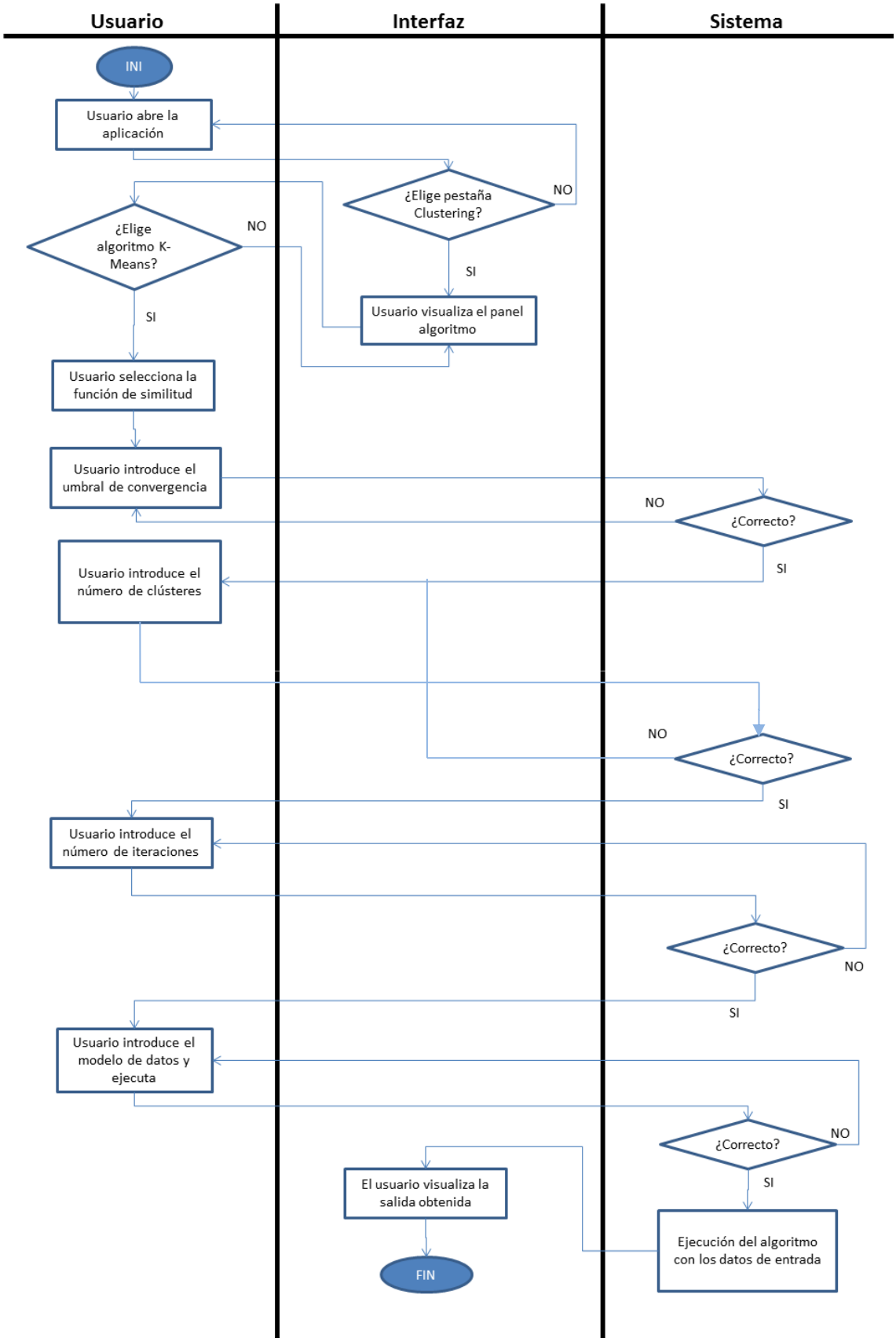


Figura 40: Diagrama de secuencia del algoritmo de clustering K-Means.

Algoritmo de clasificación Naive Bayes

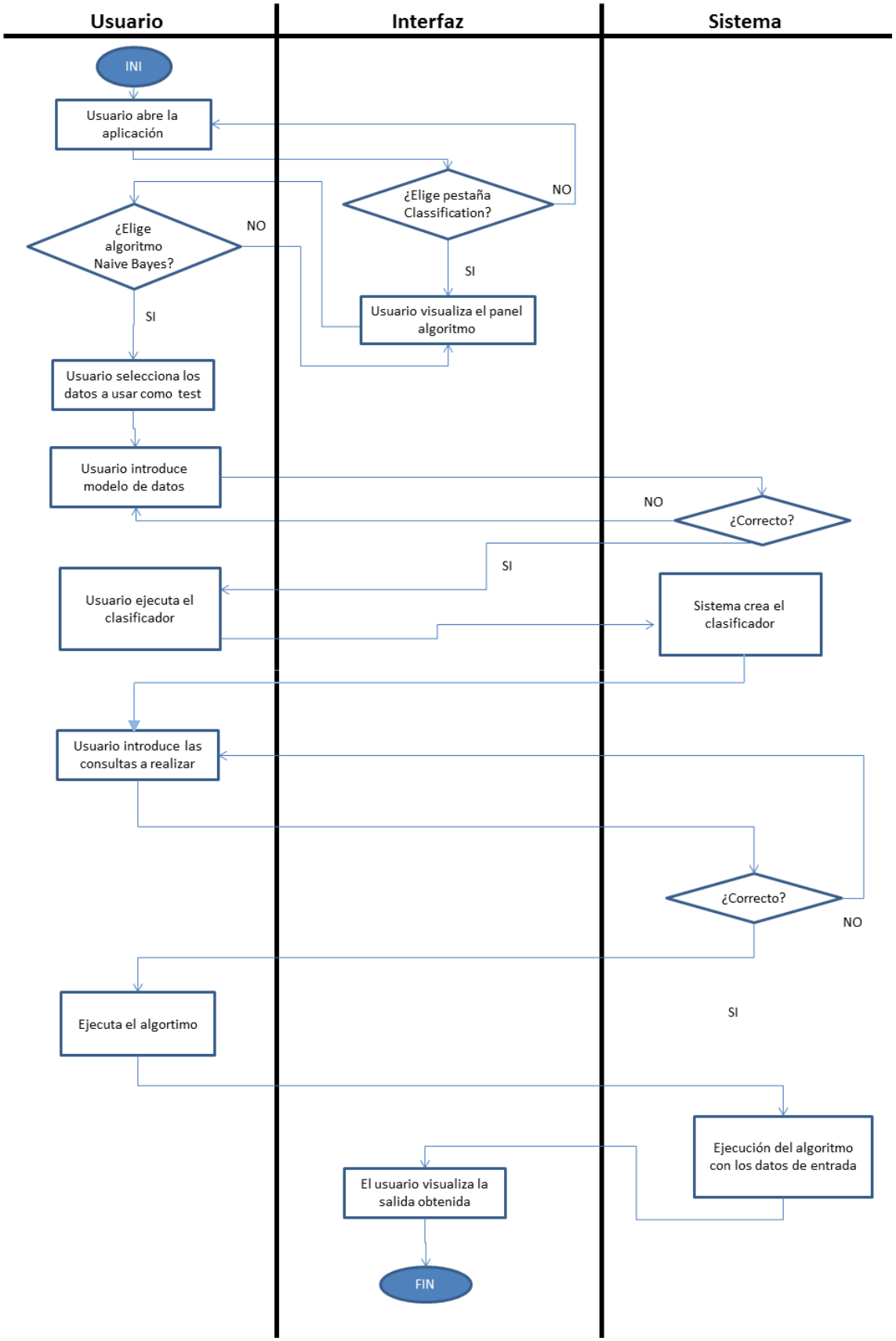


Figura 41: Diagrama de secuencia del algoritmo de clasificación Naive Bayes.

5. Implementación

5.1. Detalles de implementación

Para la implementación del proyecto hemos usado el entorno de desarrollo Eclipse IDE. Como sistema de control de versiones hemos utilizado Git haciendo uso de los servidores públicos de Github, Inc. (donde se encuentra alojado nuestro código fuente) y mediante el plugin eGit para eclipse. También nos hemos ayudado de la aplicación de escritorio gratuita SourceTree cuyo principal uso es el de gestionar de manera muy clara las distintas versiones y cambios generados a lo largo del desarrollo.

El lenguaje de programación usado mayoritariamente ha sido Java, usando las distribuciones de Oracle jdk1.7 y actualizando al final del desarrollo a la nueva versión estable de java, la 1.8. El motivo de la elección de Java como lenguaje de programación es que los frameworks Apache Mahout y Apache Hadoop están escritos en Java.

Tambien hemos usado algunas librerías gratuitas para ayudarnos en el diseño y testeo de la aplicación, por ejemplo la librería de Logs de Apache, Log4j, Ballontip para la creación de los tips emergentes con la ayuda, así como la clase DisableNode que extiende DefaultMutableTreeNode, que no ha permitido cambiar el color de los nodos del JTree cuendo este se encontraba activo.

A continuación se muestra una porción de código usado para crear el Job de Hadoop, código contenido en la clase ClusterBuilder. La finalidad última de este código es ejecutar la clase KMeansDriver, tomando los argumentos de la llamada de las demás clases de la aplicación.

```
public static void getHadoopJob() {
    Configuration confHadoop = new Configuration();
    try {
        // crear el chunk
        String[] args1 = JobClusterBuilder.buildSequenceFromDirectoryJob();
        ToolRunner.run(new SequenceFilesFromDirectory(), args1);

        // crear los vectores
        String[] args2 = JobClusterBuilder.buildSparseVectorsJob();
        ToolRunner.run(new SparseVectorsFromSequenceFiles(), args2);

        String clusterIn = DataModelClusterPanel.getOutputPath() + System.getProperty("file.separator") + "clusters";

        // crear los clusters iniciales
        CanopyDriver.run(confHadoop, new Path(args2[3] + System.getProperty("file.separator") + "tfidf-vectors"), new Path(clusterIn), d, t1, 0.9, true,
                t1, !hadoop);

        String[] args3 = JobClusterBuilder.buildSplitJob();
        ToolRunner.run(new Configuration(), new SplitInput(), args3);

        String inputPoints = args2[3] + System.getProperty("file.separator") + "tfidf-vectors";
        String initialClusters = clusterIn + System.getProperty("file.separator") + "clusters-0-final";

        String outputK = args2[3] + System.getProperty("file.separator") + "output";

        KMeansDriver.run(confHadoop, new Path(inputPoints), new Path(initialClusters), new Path(outputK), d, t1, iteraciones, true, t1, !hadoop);

        String read = outputK + System.getProperty("file.separator") + "clusters-" + "1-final" + System.getProperty("file.separator") + "part-r-00000";

        writeResultHadoop(confHadoop, read);
        MainGUI.writeResult("Hadoop Job finished with success.", Constants.Log.INFO);
    } catch (Exception e) {

        MainGUI.writeResult(e.getMessage(),Constants.Log.ERROR);
    }
}
```

La estructura de un sistema de recomendación basado en usuario (local) es la siguiente:

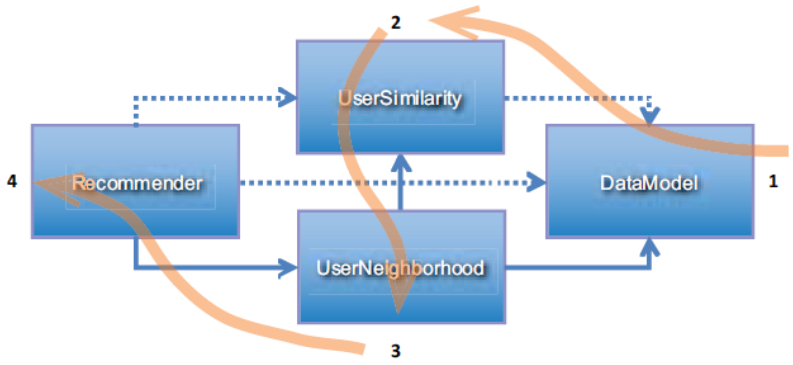


Figura 42: Sistema de recomendación basado en usuario. Ref. [27]

Como podemos observar en la porción de código Java que se muestra a continuación, construimos el objeto *Recommender* a partir del modelo de datos, la similitud de usuario y la función Neighborhood.

```
public static Recommender buildRecommender() {
    if (TypeRecommenderPanel.getSelectedType().equals(Constants.RecommType.USERBASED)) {
        DataModel model = DataModelRecommenderPanel.getDataModel();
        if (model != null) {
            UserSimilarity similarity = SimilarityRecommenderPanel.getUserSimilarity(model);
            UserNeighborhood neighborhood = NeighborhoodRecommenderPanel.getNeighborhood(similarity, model);
            return new GenericUserBasedRecommender(model, neighborhood, similarity);
        } else {
            MainGUI.writeResult("Trying to run a recommender without a dataModel loaded.", Constants.Log.ERROR);
            return null;
        }

    } else if (TypeRecommenderPanel.getSelectedType().equals(Constants.RecommType.ITEMBASED)) {
        DataModel model = DataModelRecommenderPanel.getDataModel();
        if (model != null) {
            ItemSimilarity similarity = SimilarityRecommenderPanel.getItemSimilarity(model);
            return new GenericItemBasedRecommender(model, similarity);
        } else {
            MainGUI.writeResult("Trying to run a recommender without a dataModel loaded.", Constants.Log.ERROR);
            return null;
        }
```

Aquí podemos ver el fragmento de código donde se crean y recogen los argumentos para la ejecución del test de un clasificador Naive Bayes por comandos:

```
// Args test

String[] argsTestnb1 = args[4] = new String[ARGS_TEST_NB];

i = 0;
argsTestnb1[i] = "--input";
argsTestnb1[++i] = testPath;
argsTestnb1[++i] = "--model";
argsTestnb1[++i] = modelPath;
argsTestnb1[++i] = "--output";
argsTestnb1[++i] = outputPath + "/result";
argsTestnb1[++i] = "-ow";
argsTestnb1[++i] = "--labelIndex";
argsTestnb1[++i] = labelIndexPath;
if (complementary) {
    argsTestnb1[++i] = "--testComplementary";
}
if (!hadoop) {
    argsTestnb1[++i] = "--runSequential";
}



String[] argsTestnb2 = args[5] = new String[ARGS_TEST_NB];

i = 0;
argsTestnb2[i] = "--input";
argsTestnb2[++i] = trainPath;
argsTestnb2[++i] = "--model";
argsTestnb2[++i] = modelPath;
argsTestnb2[++i] = "--output";
argsTestnb2[++i] = outputPath + "/result";
```

5.2. Instalando *easyMahout*

5.2.1. Requisitos mínimos

Tanto Apache Hadoop como Apache Mahout han sido originalmente diseñados bajo la arquitectura UNIX, por lo que para ejecutar satisfactoriamente tareas Mapreduce en Hadoop necesitaremos un PC corriendo sobre alguna plataforma UNIX.

Plataformas soportadas

- GNU/Linux
- Win32 (Tareas distribuidas no funcionan bien al 100%)

Software requerido

Tanto para Linux como para Windows:

- ORACLE Java JDK 1.7 o superior
- Apache Mahout Distribution 0.8
- Apache Hadoop 1.1.2 (o 1.x)

Requisito adicional para Windows:

- Cygwin

5.2.2. Instalando Hadoop

Nuestra distribución de easyMahout ya trae consigo la versión 1.1.2 de Hadoop. EasyMahout ha sido desarrollado basándonos en esa versión, por lo que para un funcionamiento correcto asegurado, utilice esa versión.

Aun así, para instalar una nueva versión de Hadoop basta simplemente con descomprimir el tar.gz de la distribución en cualquier directorio del sistema. (Preferiblemente en easyMahout/easyMahout1_lib/Hadoop-xxxxx)

5.2.3. Instalando Mahout

Nuestra distribución de *easyMahout* ya trae consigo la versión 0.8 de Mahout. EasyMahout ha sido desarrollado basándonos en esa versión, por lo que para un funcionamiento correcto asegurado, utilice esa versión.

Aun así, es posible instalar una versión diferente de Mahout descomprimiendo el tar.gz de la distribución en cualquier directorio del sistema. (Preferiblemente en easyMahout/easyMahout1_lib/mahout-distribution-0.x).

5.2.4. Preconfiguración de las variables de entorno

Tanto si ha decidido cambiar de versión de Mahout o Hadoop, debemos configurar la variable de entorno JAVA_HOME de la siguiente manera:

File -> Preferences ->

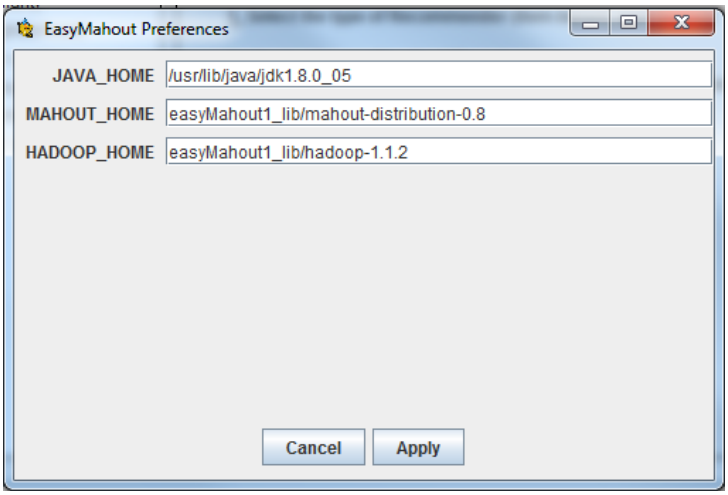


Figura 43: Ventana de preferencias de easyMahout.

Esta variable debe contener el directorio raíz de vuestra instalación de Java. Esta parte es muy importante para poder ejecutar correctamente Map-Reduce Jobs sobre Linux. Las variables MAHOUT_HOME y HADOOP_HOME estas configuradas por defecto. Estos parámetros se guardan en un archivo XML ubicado en easyMahout1_lib/preferences.xml, de modo que solo es necesario configurarlo la primera vez.

5.2.5. Tabla de compatibilidad

Algoritmo	Windows		GNU/Linux	
	Local	Distribuido	Local	Distribuido
Recom. basado en usuario	x	n/a	x	n/a
Recom. basado en objeto	x	n/a	x	x
Factorización de matrices	x	n/a	x	x
Clustering Canopy	x	x	x	x
Clustering K-Means	x	x	x	x
Clustering Fuzzy K-Means	x	x	x	X
Clasificacion Naive Bayes	n/a	n/a	n/a	X

Los algoritmos marcados como **n/a** en Windows no son funcionales en su totalidad debido a que Hadoop ha sido desarrollado sobre una arquitectura UNIX.

Los algoritmos marcados como **n/a** en GNU/Linux no han sido implementados por la comunidad de Apache Mahout.

5.3. Uso general de la aplicación

Para arrancar la aplicación, debemos seguir pasos distintos según nuestro Sistema Operativo sea Windows o Linux. En Windows simplemente basta con ejecutar el JAR de easyMahout e inmediatamente aparecerá la ventana principal.

En Linux, tenemos dos opciones para correr la aplicación. Cualquier usuario con un mínimo de experiencia en SO Linux y Java habrá pensado: Abrir el terminal y ejecutar el programa mediante el comando *java –jar easyMahout1.0.jar.* Correcto, los pasos serían los siguientes.

1. Abrir el terminal
2. Situarnos en el directorio donde se encuentra el JAR de easyMahout y la carpeta con las librerías dependientes
3. Lanzar el comando *java –jar easyMahout1.0.jar*

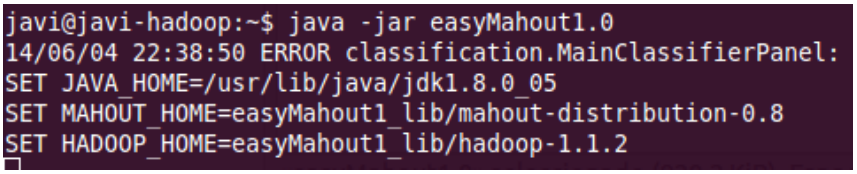


Figura 44: Ejecución a través del terminal.

La ventaja de esta opción es que podremos ver en el terminal los logs internos de las tareas mapreduce de Mahout. Puede ser útil como curiosidad o como modo de depuración.

La otra manera es incluso más sencilla:

1. Click derecho sobre el JAR de easyMahout, pulsar sobre “Abrir con…”
2. Pinchar sobre “Ejecutar un comando externo” y escribir *java –jar*
3. Click en Abrir

A partir de este momento, el Sistema Operativo recordará esta acción, y siempre que hagamos doble click sobre el JAR de easyMahout, lo abrirácon Java.

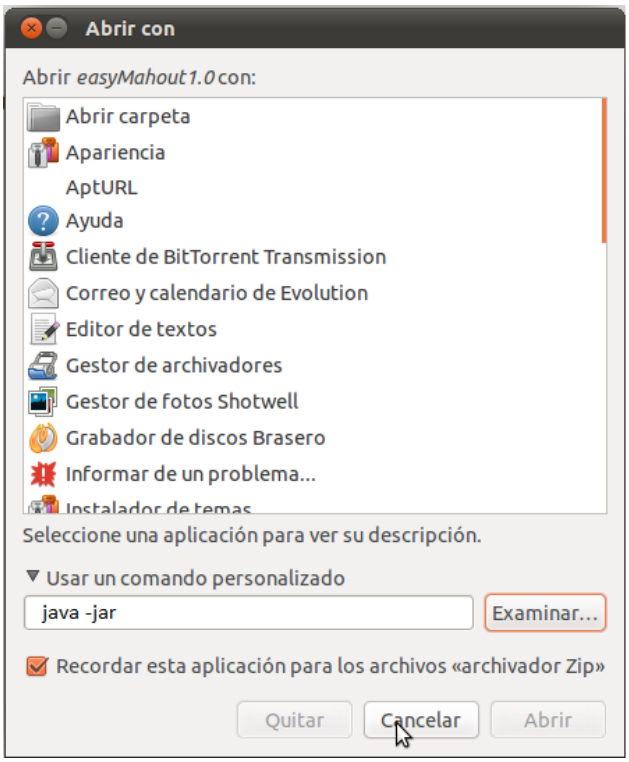


Figura 45: Ejecutar easyMahout desde el escritorio.

La aplicación easyMahout ha sido concebida con el propósito primordial de ser una aplicación fácil, simple y sencilla, pero con una funcionalidad muy potente. Gracias a tomarnos este objetivo muy en serio, podemos construir un sistema de recomendación, clasificador o agrupador en tan solo unos minutos o incluso segundos.

El sistema de navegación por la aplicación es muy intuitivo, además la propia aplicación sirve de guía sobre los algoritmos y parámetros a introducir.

El primer paso que tenemos que dar es seleccionar el tipo de ejecución (secuencial o mapreduce) que queremos que tome nuestro algoritmo. Éstos están implementados de manera completamente diferente según el modo de ejecución, por lo que tendremos que configurar distintos parámetros en cada caso. Algunos algoritmos, como es el caso de *user-based recommender* no están soportados en la versión mapreduce, pero no se preocupe, según cambie entre el modo local y distribuido, los algoritmos elegibles variarán dinámicamente.

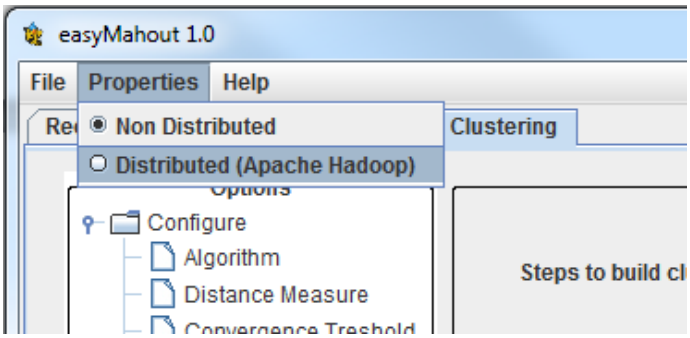


Figura 46: Elección modo de ejecución.

Cambiar entre tipos de algoritmos es tan sencillo con pulsar sobre la pestaña que desee. Una vez situados en la pestaña deseada, el primer paso en todos los casos será seleccionar el algoritmo a usar, pues de él dependen el resto de configuraciones. Según el algoritmo elegido, algunas opciones dentro del árbol de opciones quedarán desactivas y otras serán activadas. Para una configuración de parámetros correcta y fácil, basta con ir completando en orden descendente las ventanas de opciones activas que se muestran en el árbol.

Si nos quedamos atascados en alguna ventana de configuración (porque por ejemplo no sabemos qué significa un parámetro, o cuál debería ser su valor apropiado), siempre podremos pulsar en el icono de ayuda que nos mostrará un bocadillo con la ayuda necesaria. (A los más mayores, os podría recordar en cierta manera nuestro ayudante CLIP de las versiones antiguas de MS Office).

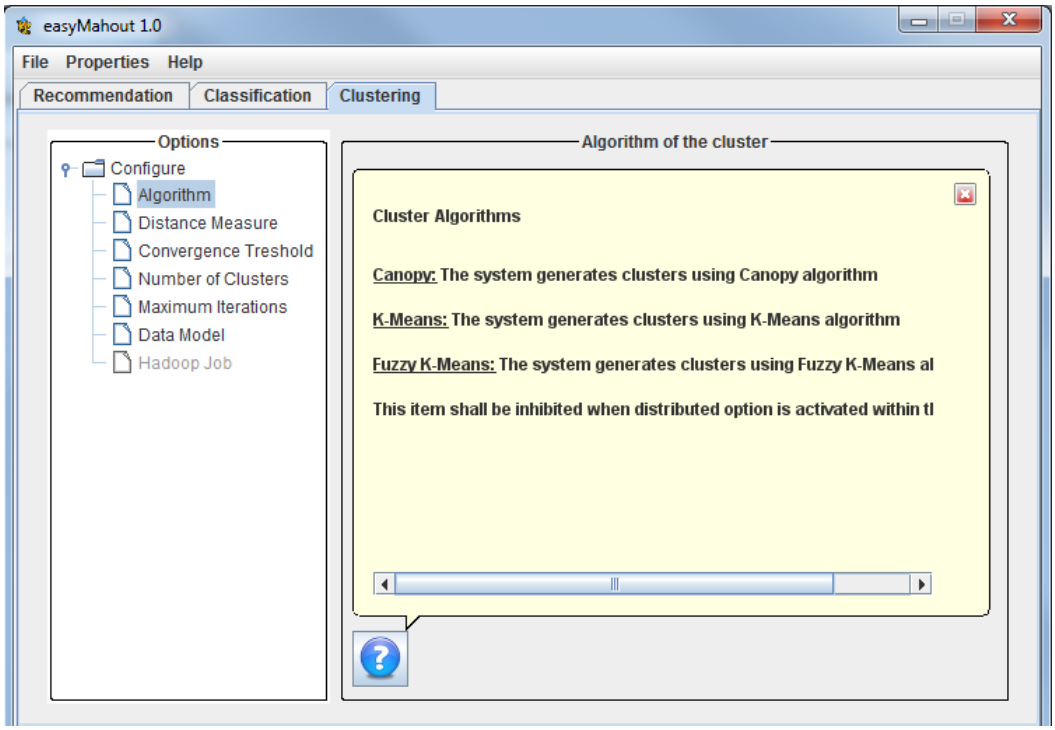


Figura 47: Vista general de la aplicación mostrando la ayuda.

La principal manera de comunicarnos con el usuario es mediante nuestro panel de Logs, el cual nos mostrará en tiempo de ejecución mensajes de información, de error y los resultados producidos por la ejecución de los algoritmos. Puede sernos muy útil para ver los errores que tememos durante la configuración del algoritmo o para saber el motivo por el cual fallo la ejecución. También es fundamental para ver los resultados, aunque ya que con easyMahout podemos tratar con ingentes cantidades de datos, podemos producir también gran cantidad de resultados. Es por ello que podremos guardar el panel de log en formato HTML o TXT para un posterior o más cómodo estudio de los resultados.

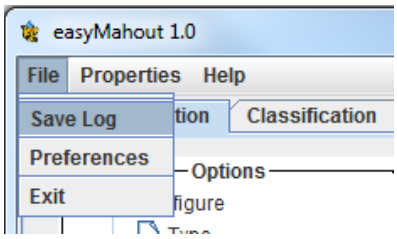


Figura 48: "Save Log" en el menú File.

La ejecución distribuida producirá siempre, además del resultado mostrado en el panel de log, archivos de salida en la ruta indicada. Estos archivos serán archivos secuenciales especiales de hadoop.

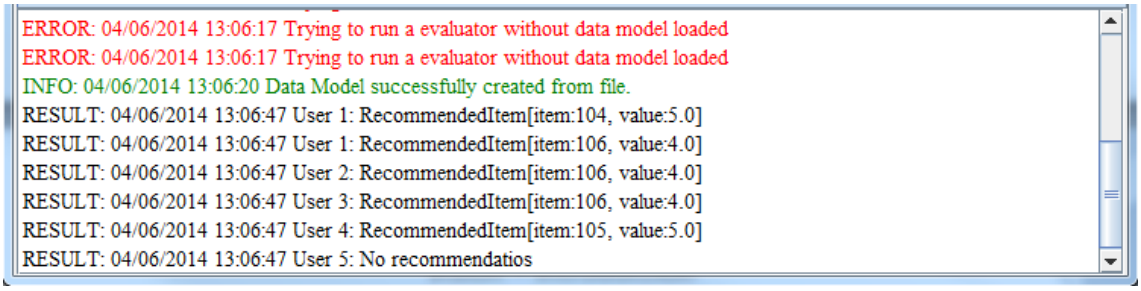


Figura 49: Panel de logs; errores, info y resultados.

6. Validación

6.1. Guía del usuario

6.1.1. Ejecución de un sistema de recomendación

Como ya vimos en la sección anterior, debemos lanzar la aplicación de una de las dos maneras propuesta, para acto seguido situarnos sobre la pestaña “Recommendation”. Para el ejemplo que vamos a desarrollar vamos a trabajar con un sistema de recomendación local, ya que tiene más pasos de configuración. La aplicación selecciona por defecto la ejecución local.

La primera ventana visible nos muestra una breve explicación de cómo construir un sistema de recomendación. Como veréis a continuación, construir un motor de recomendación es tan sencillo como seguir esos simples seis pasos.

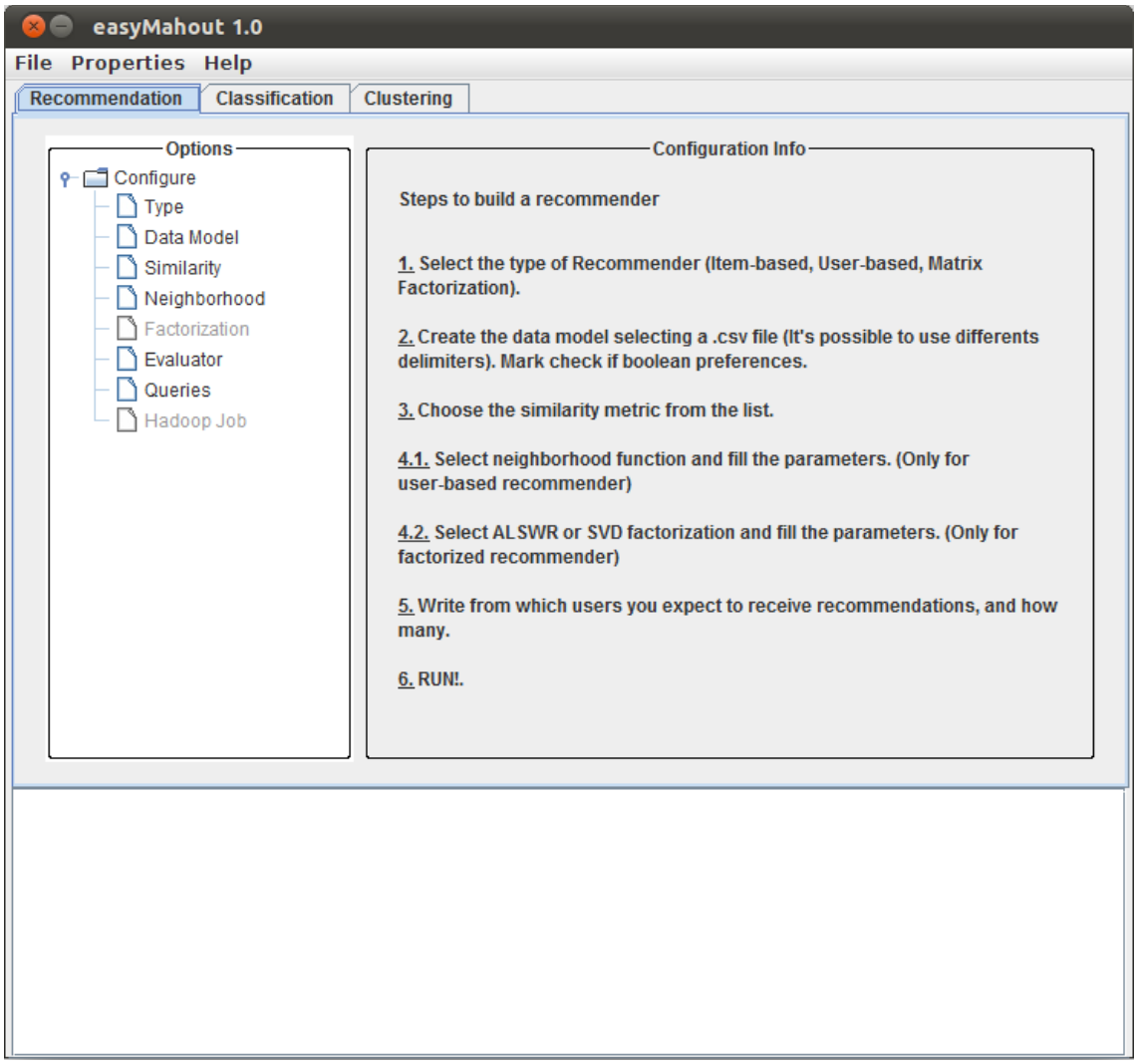


Figura 50: Flujo ejecución del sistema de recomendación, ventana principal.

El siguiente para será seleccionar que tipo de sistema de recomendación deseamos construir. Para la ejecución local hay implementados tres algoritmos, user-based, ítem-based y matrix factorization. Para nuestro ejemplo, seleccionaremos Matrix Factorization Recommender, como vemos en la Figura 19.

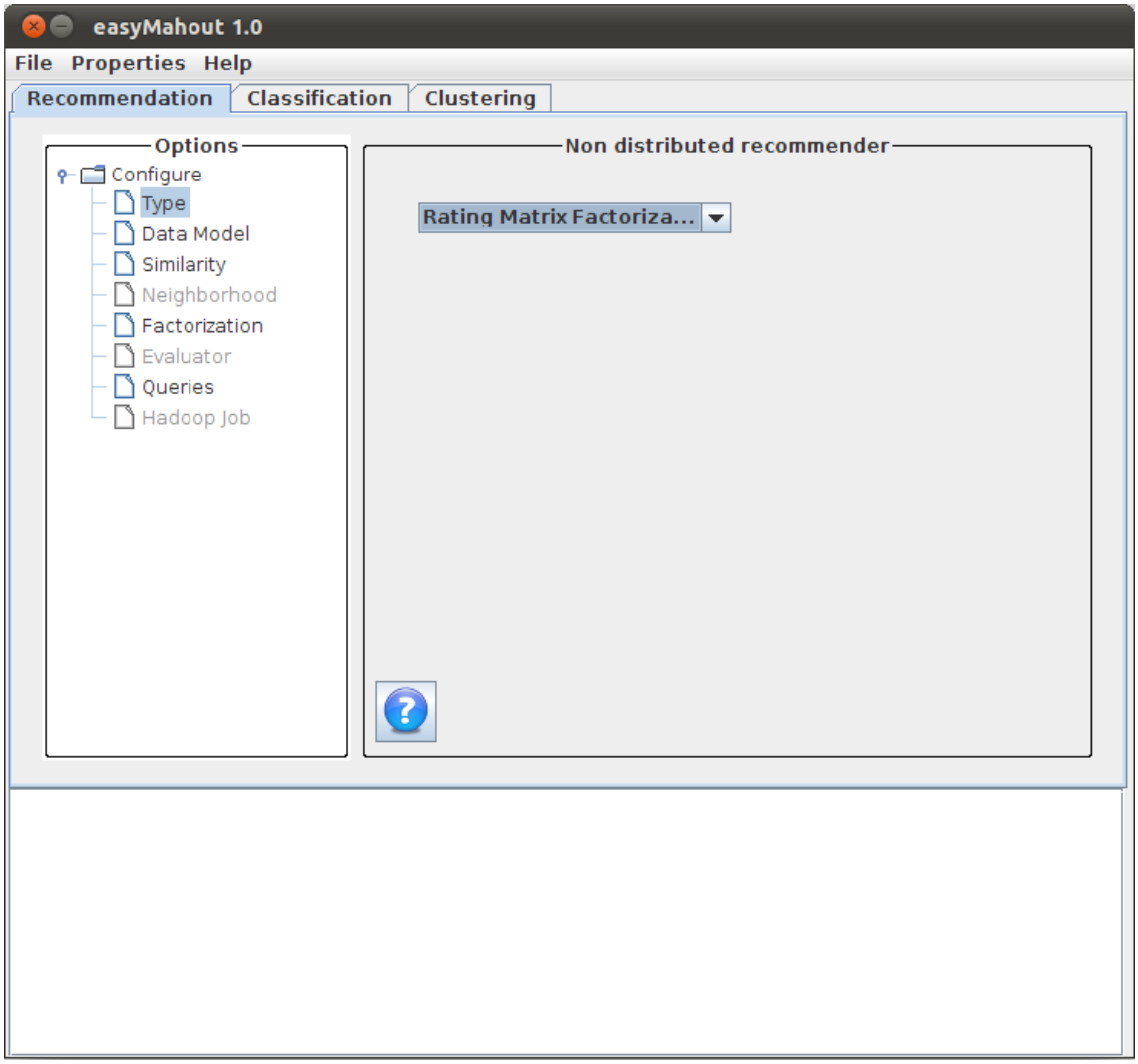


Figura 51: Flujo ejecución del sistema de recomendación, selección del algoritmo a ejecutar.

Si hemos estado atentos, habremos notado que las opciones “Neighborhood” y “Evaluator” han sido desactivadas dinámicamente, pues nuestro algoritmo no precisa de esas funciones. Además, la opción “Factorization” aparece ahora disponible, y más adelante tendremos que configurarla.

Nos situamos en la sección “Data Model” donde tenemos que elegir el fichero de datos de entrada. Recordamos que la entrada debe ser un fichero CSV (Comma Separated Values), o derivados, separados por distintos delimitadores como podrían ser: { ; : :: . }. Si nuestro delimitador no es la coma, debemos cambiar el modelo de datos de Generic a Extended, y propocionar a la aplicación en tipo de delimitador. Por ultimo faltaría decir que también tenemos la posibilidad de que las valoraciones de usuario no sean una puntuación numérica, sino simplemente una marca de si le gusta o no. Una vez hemos seleccionado el archivo, pulsamos en el botón “Create Model”, que construirá el modelo de datos a partir del fichero de entrada, indicándonos por el log de la aplicación si todo fue bien o en su defecto algo falló durante el proceso.

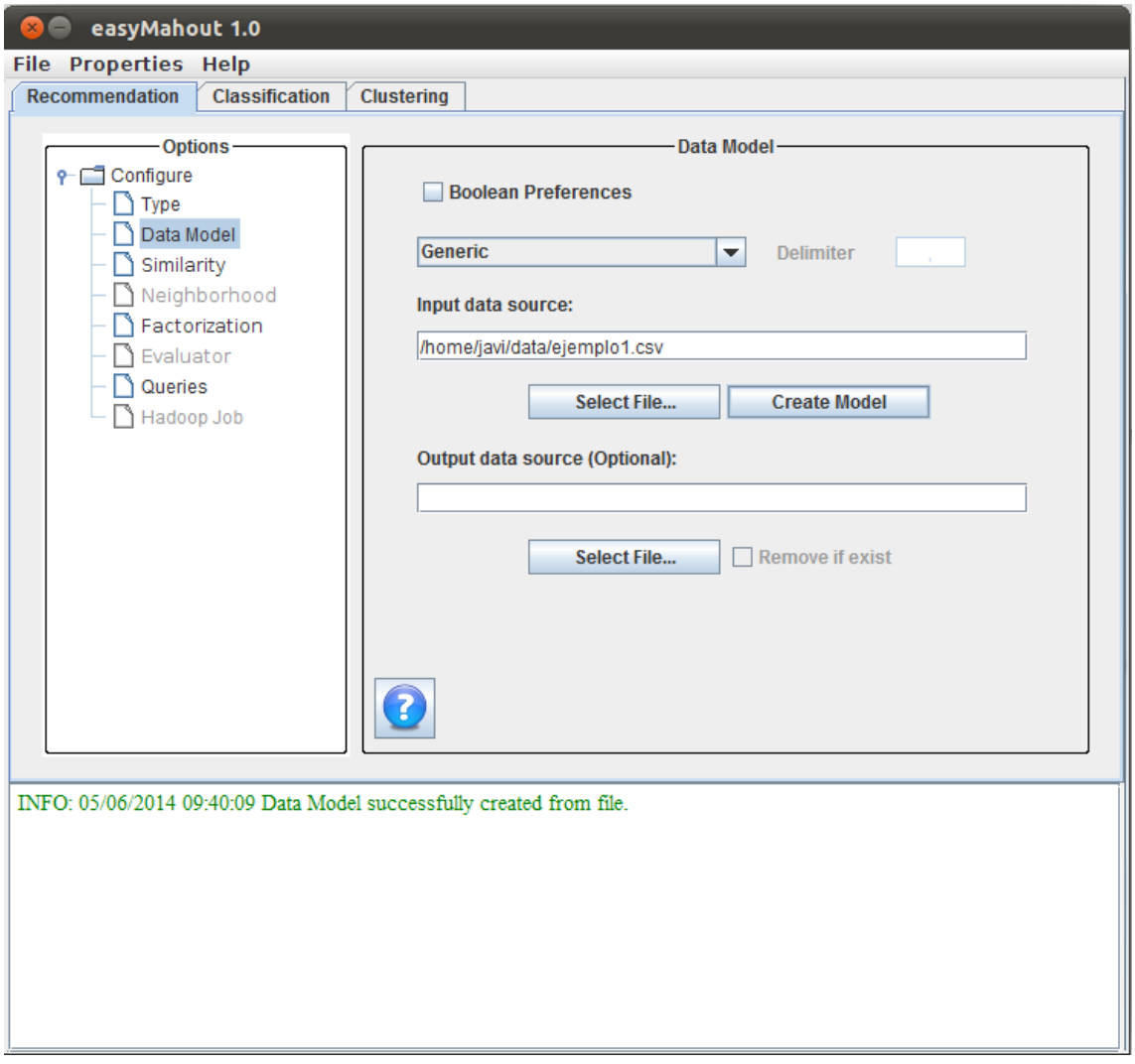


Figura 52: Flujo ejecución del sistema de recomendación, creando el modelo de datos.

A continuación debemos elegir la métrica de similitud dentro de las implementaciones posibles ofrecidas por Mahout, y dependiendo de la métrica elegida, podremos ponderar los resultados de la correlación en función del número de valoraciones del objeto. Como ya vimos en la introducción a la recomendación, esto puede sernos útil para mitigar los posibles problemas de la correlación de Pearson.

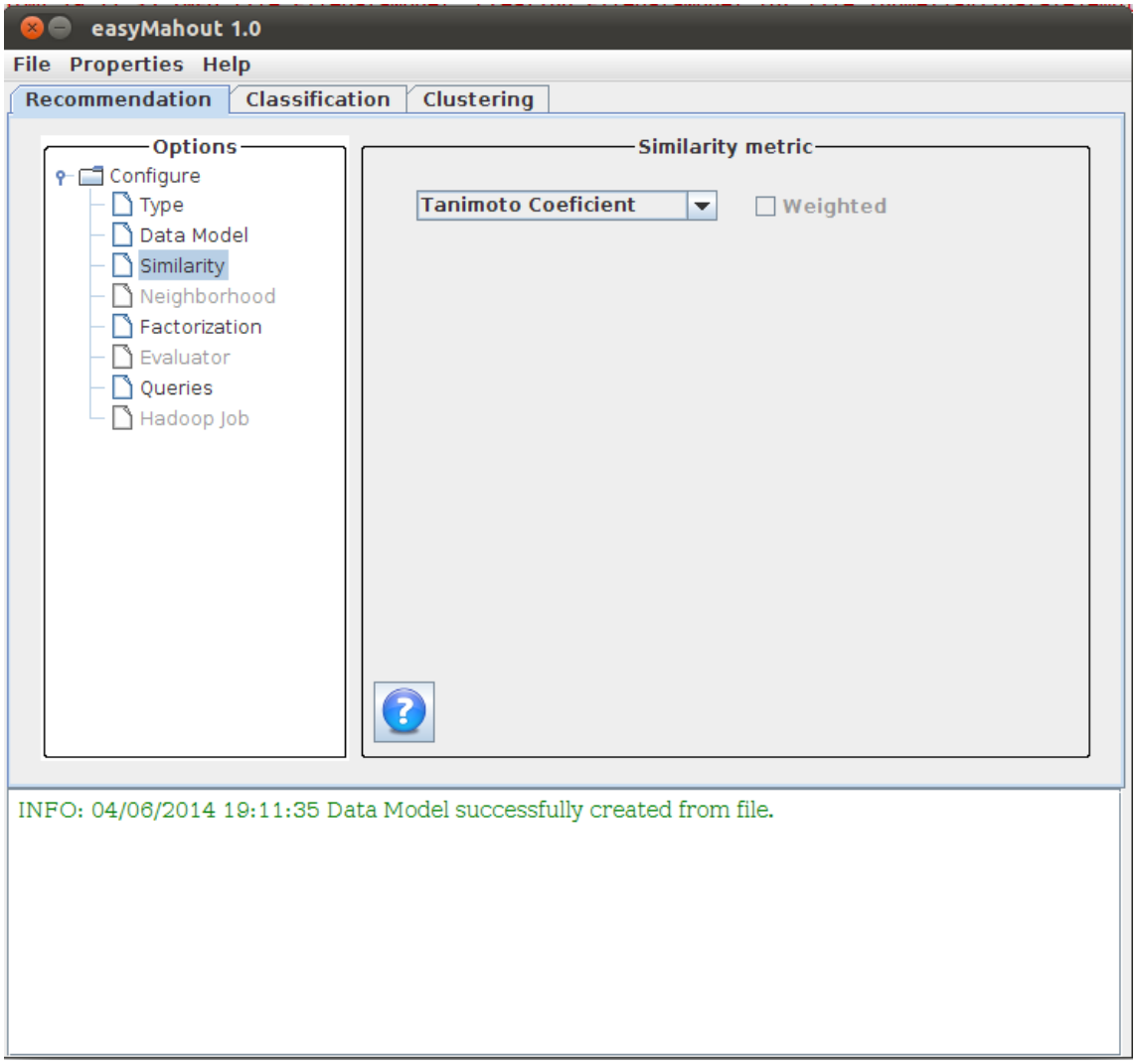


Figura 53: Flujo ejecución del sistema de recomendación, seleccionando métrica de similitud.

Pasamos a la siguiente ventana activa del árbol de opciones, en este caso "Factorization". Primero decidimos como vamos a factorizar la matriz de valoraciones en las matrices U y M, mediante el método "Alternating Least Squares" o "Singular Value Decomposition". Una vez seleccionado el método, puelsamos sobre el botón configurar.

En esta nueva ventana emergente, debemos seleccionar el número de características por usuario a estudiar, el número de iteraciones que realizara el algoritmo de factorización y el factor lambda, que como vimos en la sección de introducción, los autores del paper de Netflix determinaron el valor 0.065 como óptimo.

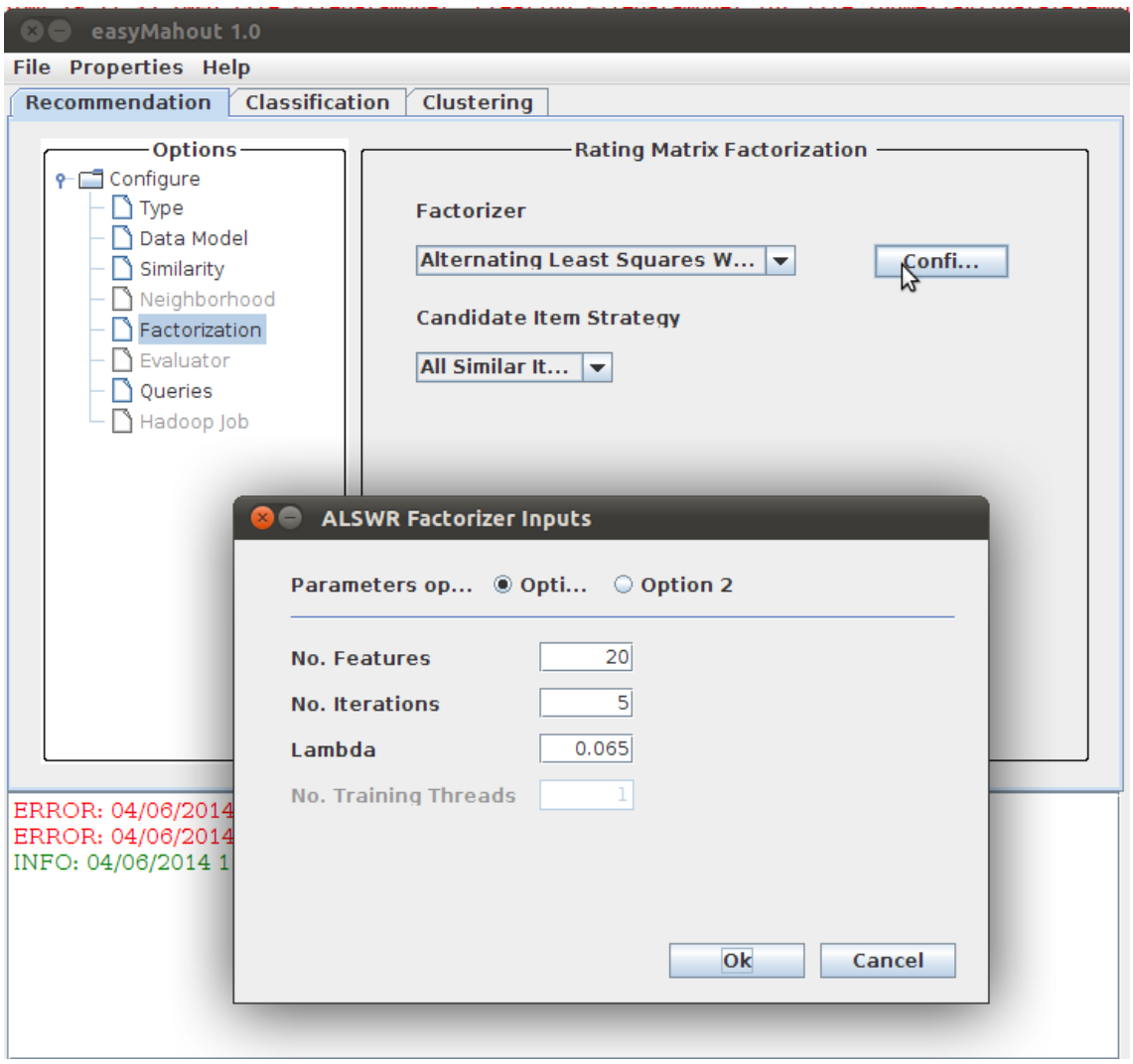


Figura 54: Flujo ejecución del sistema de recomendación, configurando mecanismo de factorización.

A diferencia de los sistema de recomendación sobre mapreduce, los cuales calculan todas las predicciones de recomendación de todos los usuarios, la versión local nos permite decidir para qué usuarios queremos generar recomendaciones, y cuantas.

Por este motivo, en la ventana “Queries” podemos decidir esta información por medio de una tabla, añadiendo una fila con el identificador de usuario y el número de recomendaciones a producir.

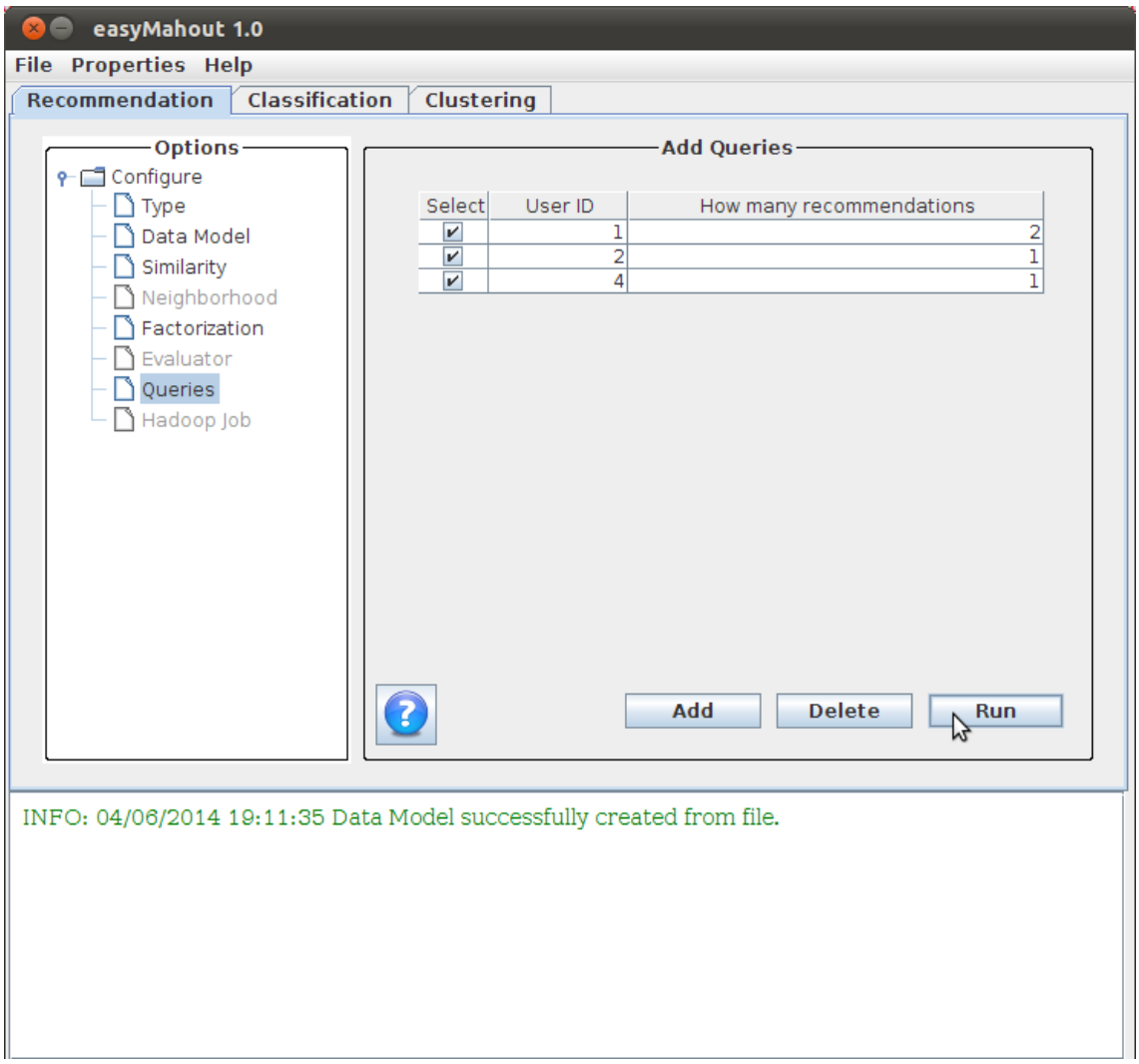


Figura 55: Flujo ejecución del sistema de recomendación, ventana queries.

Llegados a este punto, con el sistema de recomendación configurado correctamente, solo queda pulsar el botón “Run” y esperar el resultado.

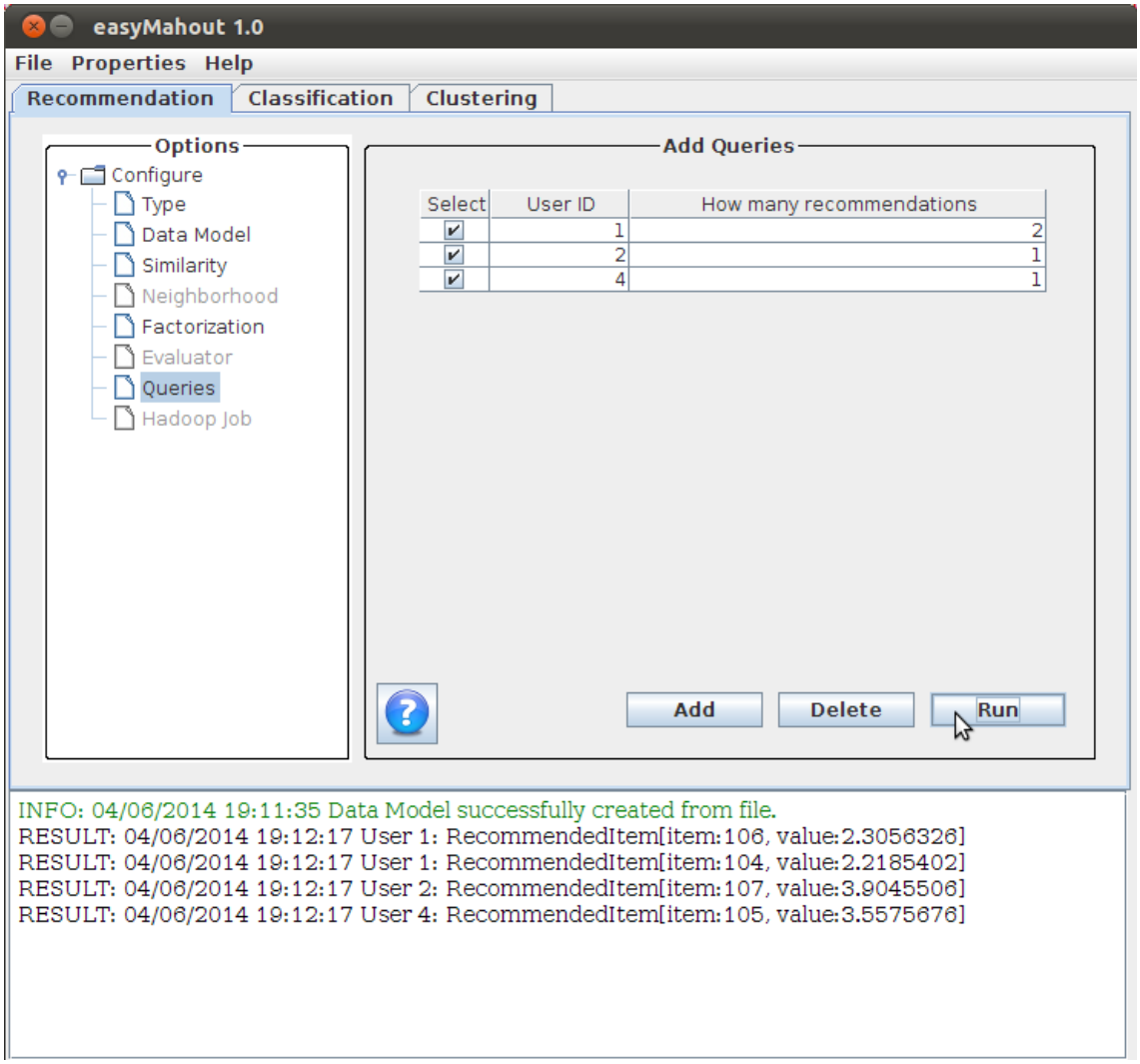


Figura 56: Flujo ejecución del sistema de recomendación, resultado.

Tal y como hicimos las “queries”, el sistema nos ha devuelto dos recomendaciones para el usuario 1, una recomendación para el usuario 2, y otra recomendación para el usuario 3.

6.1.2. Ejecución de un algoritmo de clustering

Vamos a explicar en detalle el proceso de ejecución para formar clústeres con los datos que tenemos. Primero ejecutamos la aplicación y seleccionamos la pestaña Clustering en la barra superior:

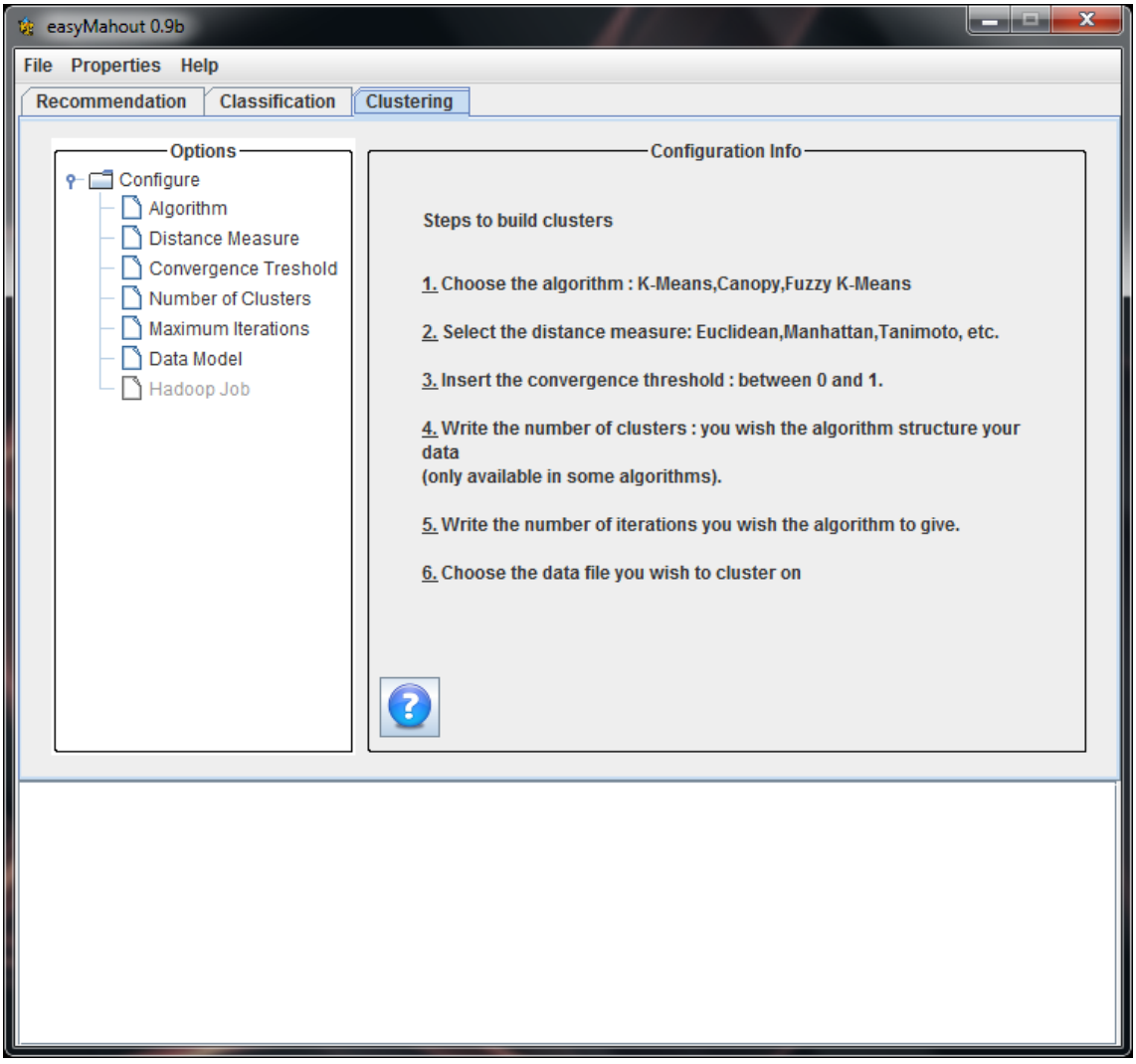


Figura 57: Flujo ejecución clustering, arranque aplicación.

A continuación seleccionamos el nivel Algorithm de la parte izquierda de la interfaz para seleccionar el algoritmo que queremos ejecutar:

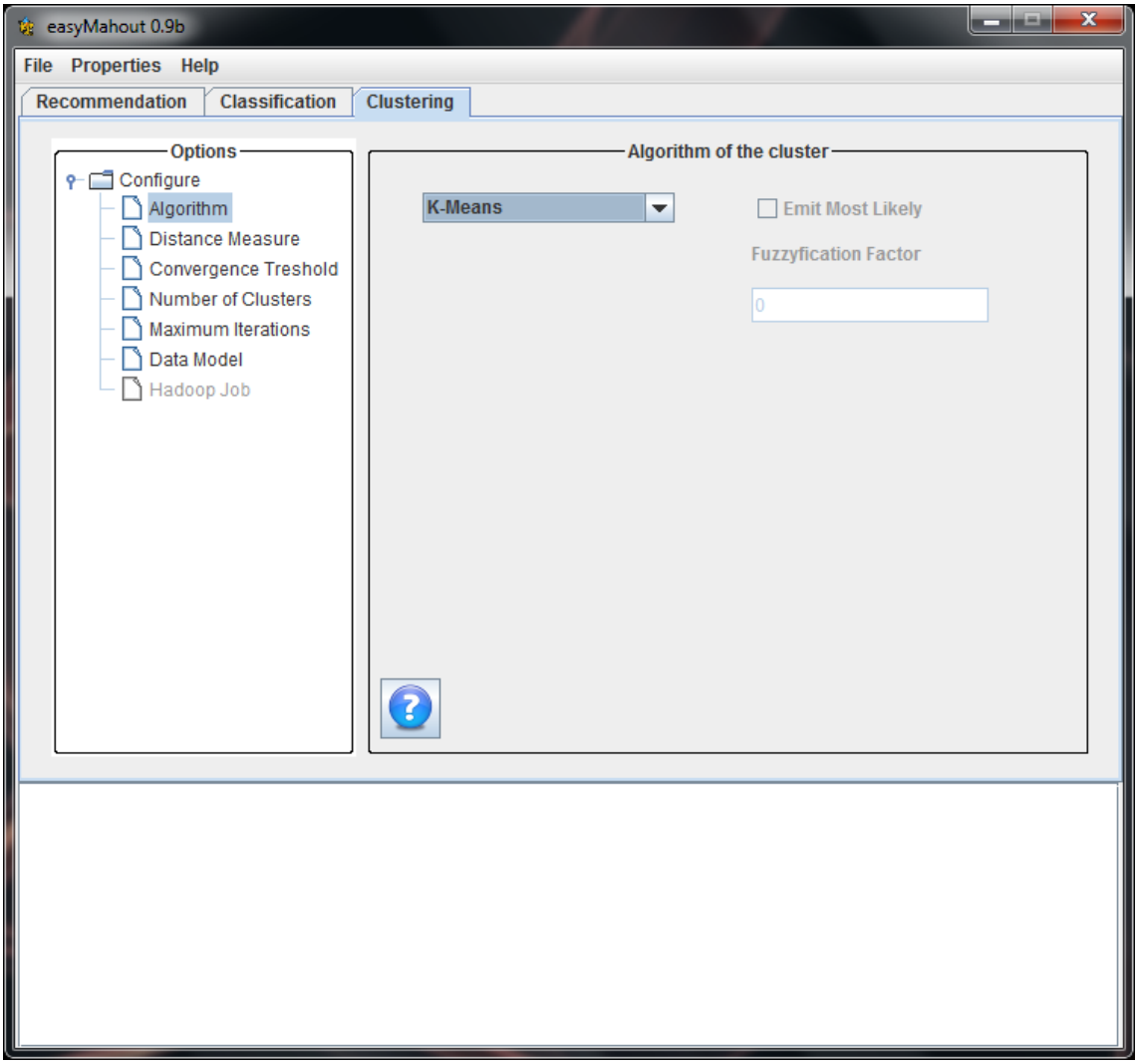


Figura 58: Flujo ejecución clustering, elección del algoritmo.

Después seleccionamos la medida de similitud:

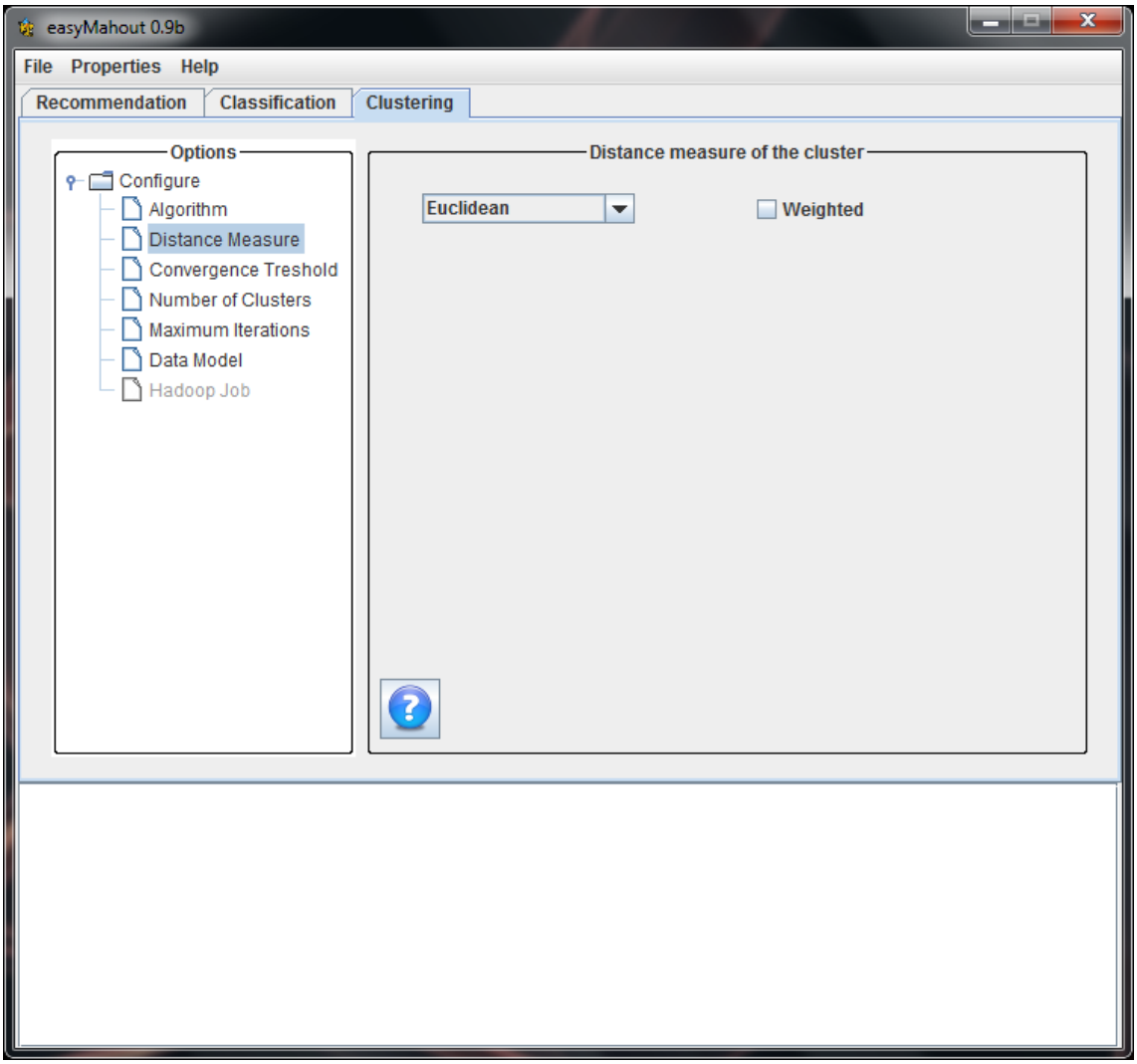


Figura 59: Flujo ejecución clustering, medida de similitud.

A continuación insertamos el umbral de convergencia:

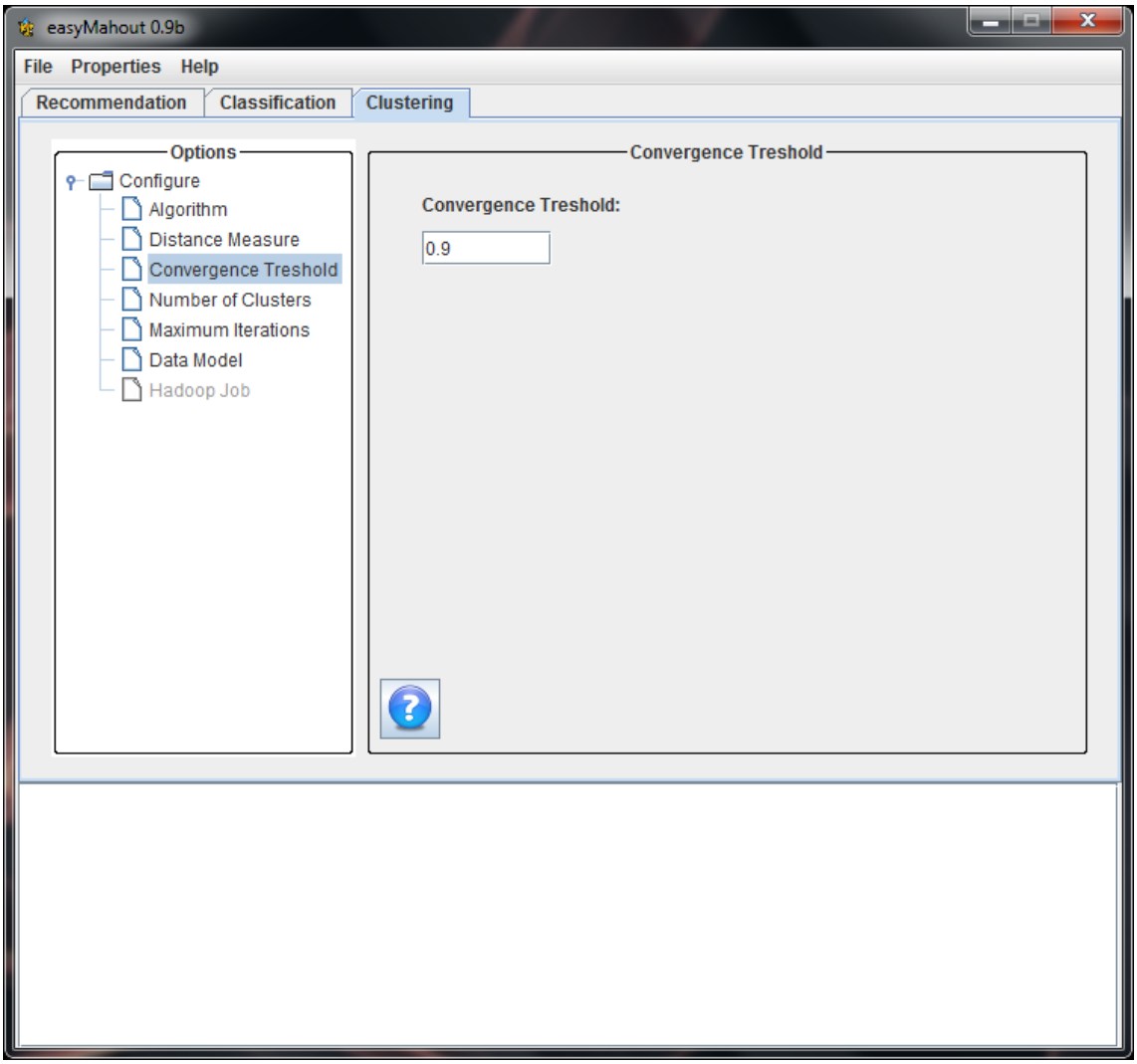


Figura 60: Flujo ejecución clustering, umbral de convergencia.

Luego, insertamos el número de clústeres que queremos que tenga la salida:

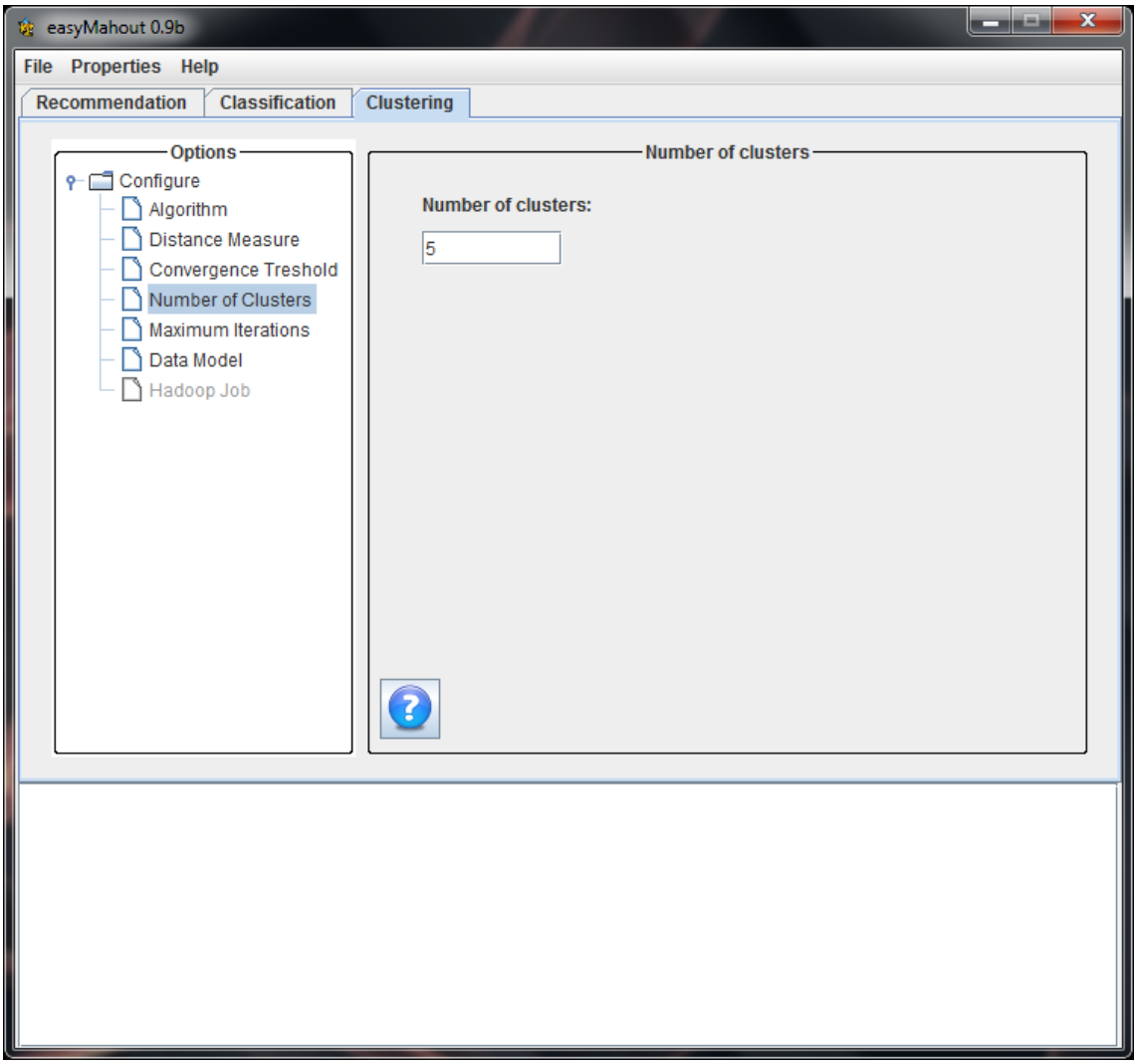


Figura 61: Flujo ejecución clustering, numero de clusters.

Después seleccionamos el número de iteraciones que queremos que el algoritmo dé:

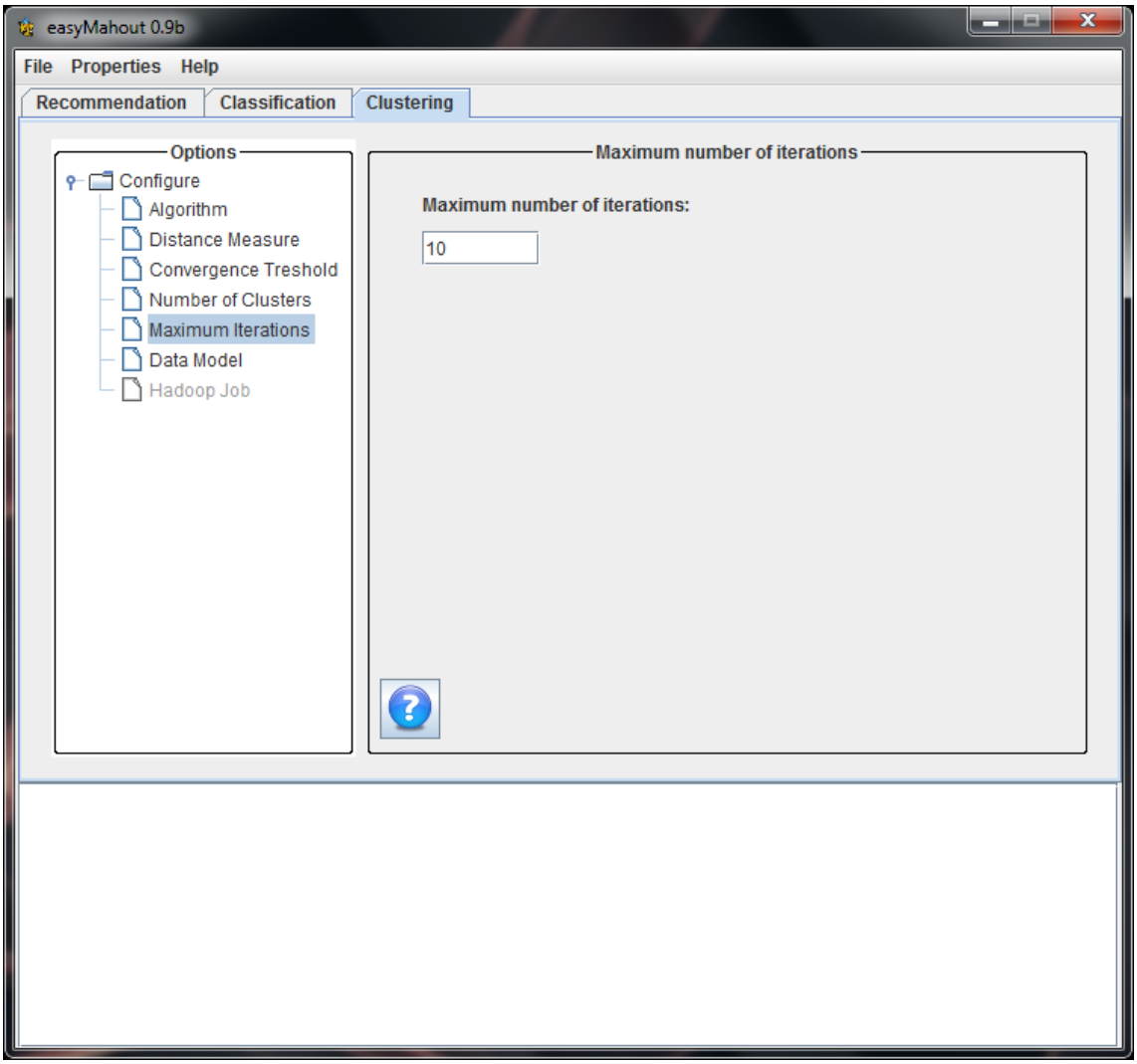


Figura 62: Flujo ejecución clustering, iteraciones.

El siguiente paso es elegir el fichero de entrada y la ruta de la salida, así como también el delimitador usado en el fichero de entrada para separar los datos:

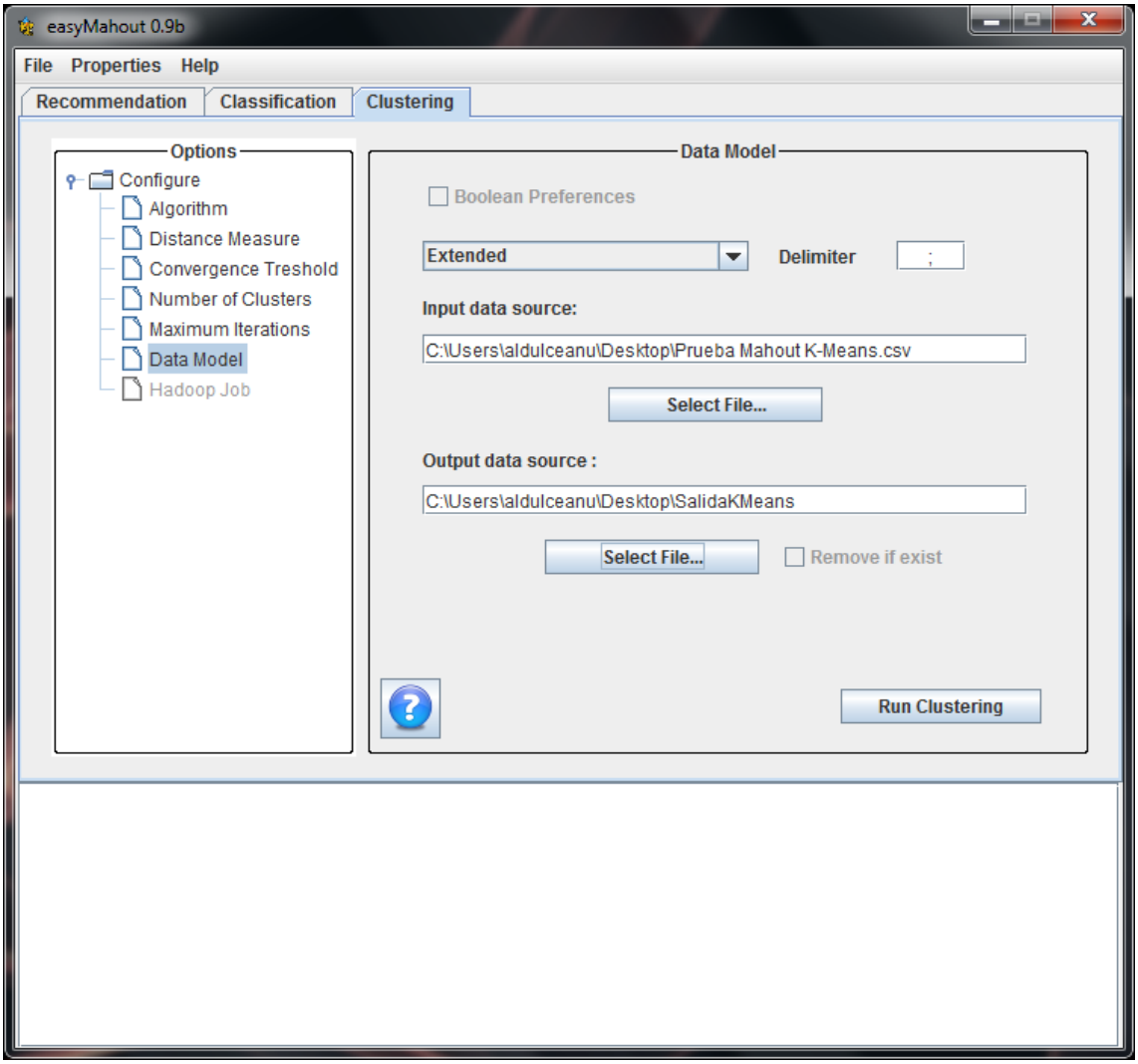


Figura 63: Flujo ejecución clustering, modelo de datos.

Por último, pulsamos el botón Run Clustering y esperamos hasta que la parte inferior de la interfaz se carga con la salida de nuestra ejecución:

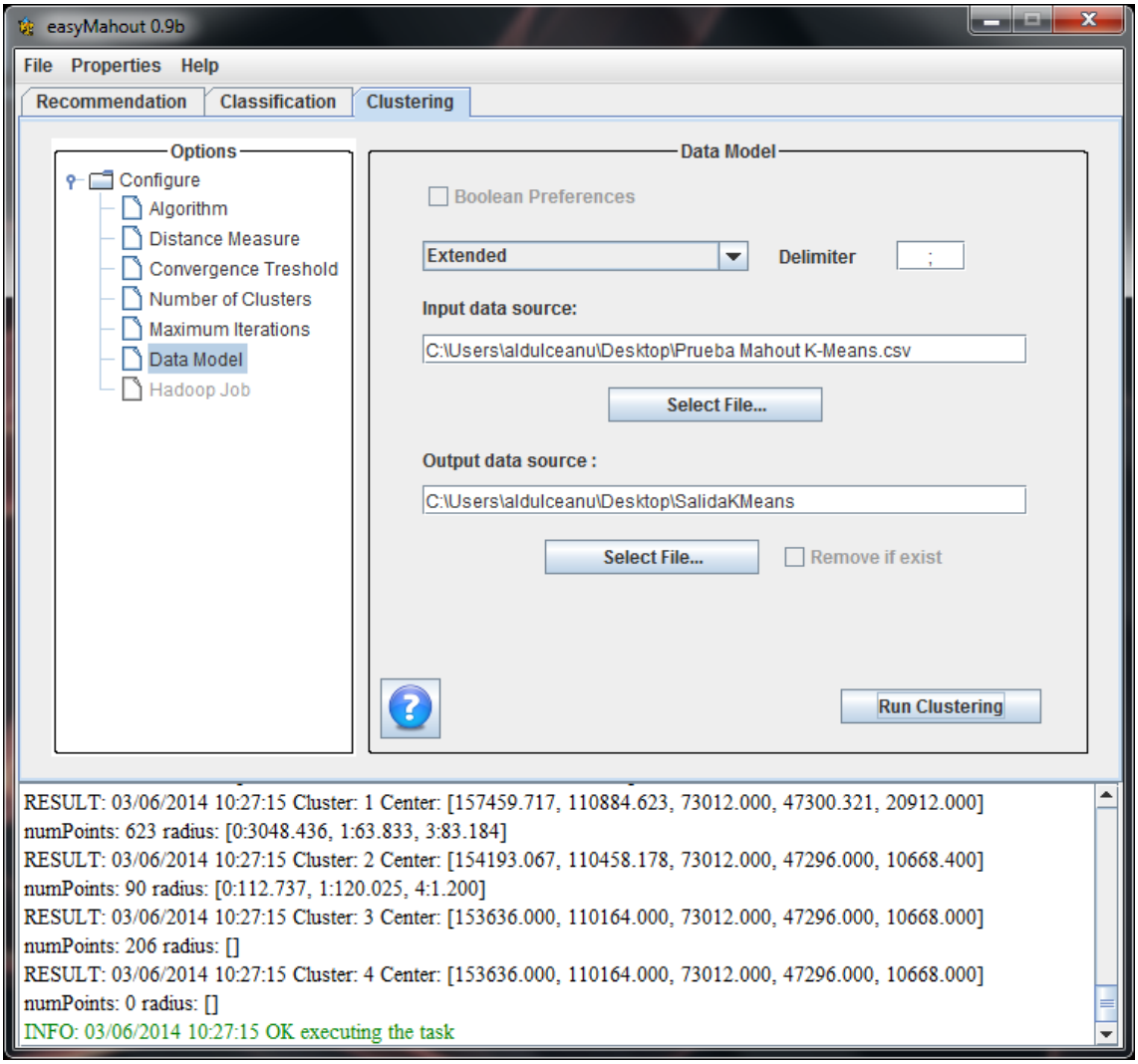


Figura 64: Flujo ejecución clustering, resultado.

6.1.3. Ejecución de un algoritmo de Clasificación

En esta sección explicaremos lo necesario para llegar a construir un clasificador, los datos requeridos, así como una explicación paso a paso del proceso. Los algoritmos disponibles son *Naive Bayes* y *Complementary Naive Bayes* para datos distribuidos (Hadoop).

Habría que empezar seleccionando la opción *Distributed* en la pestaña *Properties* del menú superior ya que los algoritmos funcionan solo de esta manera.

Después tendríamos que seleccionar el algoritmo a utilizar, nótese que los paneles *Data Definitions* y *Training Data* solo están disponibles para el algoritmo *SGD* el cual no está en producción.

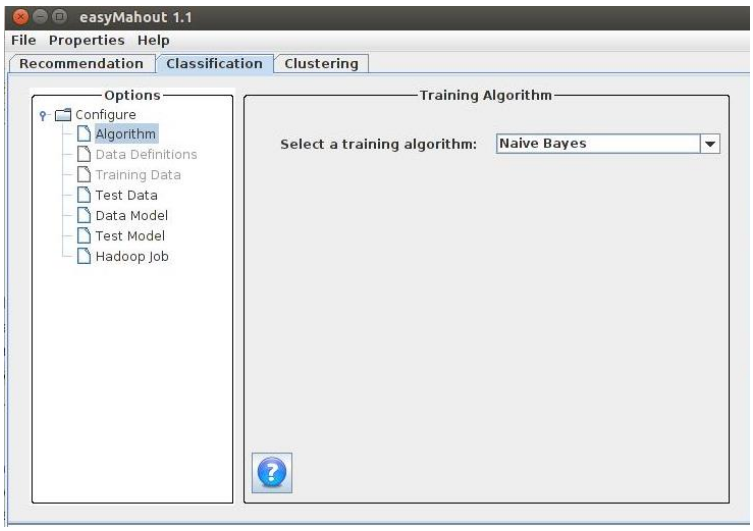


Figura 65: Flujo ejecución del sistema de clasificación, ventana algoritmo.

Lo siguiente sería elegir que parte de los datos de la entrada que van a ser utilizados para testear al modelo creado durante el entrenamiento, decir también que luego se podrá testear el modelo creado a parte.

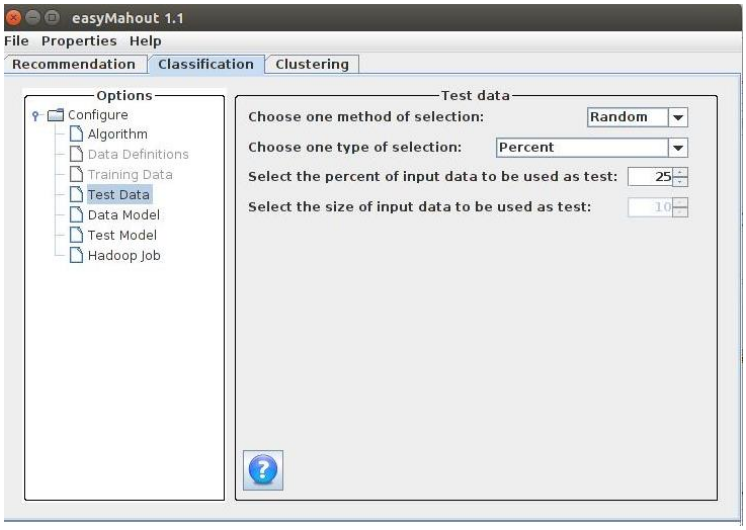


Figura 66: Flujo ejecución del sistema de clasificación, ventana test data.

Aquí podemos seleccionar un porcentaje de los datos de entrada o un tamaño fijo de los datos, dependiendo de las opciones de creación unas estarán disponibles y otras no. Aparte de que la selección sea aleatoria o un rango determinado por el usuario.

Después pasaríamos al panel *Data Model* donde indicaríamos los datos de entrada y la ruta donde queremos que se guarde la salida generada por el algoritmo. Una vez seleccionados los directorios habría que pulsar el botón *Run Classifier* para que empiece la ejecución. El resultado obtenido debería parecerse a lo siguiente:

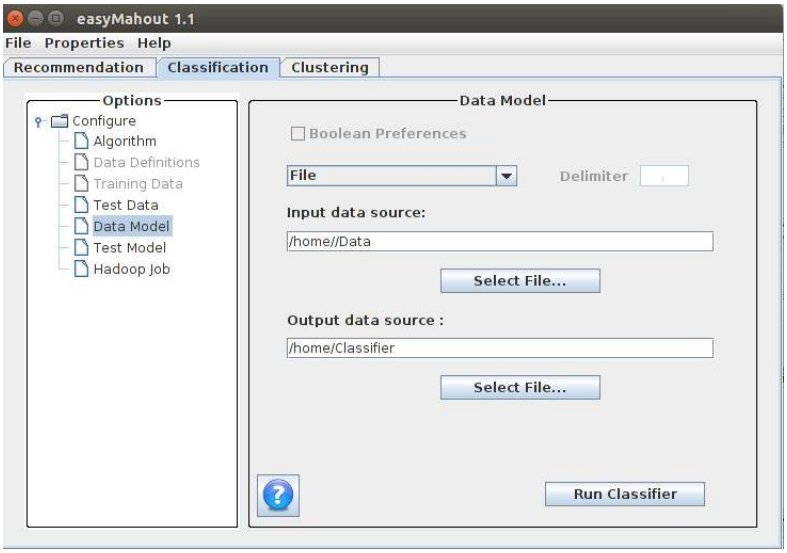


Figura 67: Flujo ejecución del sistema de clasificación, ventana Model data.

La salida obtenida se parecerá a la siguiente:

Figura 68: Flujo ejecución del sistema de clasificación, salida mostrada.

Es posible testear un modelo ya creado durante un entrenamiento con nuevos datos o con los mismos para probarlo. Para ello debemos disponer de un modelo y un índice de etiquetas (*label index*) creado durante un entrenamiento previo. En el panel *Test Model* introducimos las rutas requeridas y pulsando *Test Classifier* empezará la ejecución del *testing* sobre el modelo introducido como entrada. La salida producida será igual que al crear el clasificador. Podemos ver aquí el panel *Test Model:*

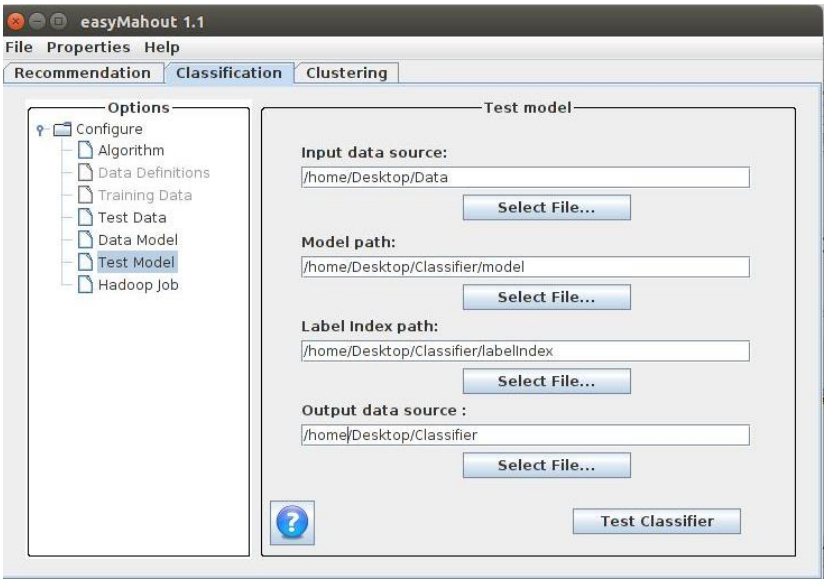


Figura 69: Flujo ejecución del sistema de clasidicación, ventana Test Model.

7. Conclusiones y trabajo futuro

7.1. Conclusiones

Los conocimientos adquiridos al diseñar y construir el sistema nos han servido para iniciarnos en lo que se considera el camino a seguir por los profesionales de la informática en el futuro próximo. Nos referimos aquí a conceptos tales como sistema de recomendación, clustering y clasificación de datos sobre la Big Data. Opinamos que nuestro trabajo será debidamente reconocido por las personas que, hasta ahora, usaban la antigua forma de recomendación, clustering y clasificación contenidos en la máquina de aprendizaje Mahout o ejecutando los algoritmos sobre el framework de aplicaciones distribuidas Hadoop, mediante comandos SHELL. Tener una interfaz gráfica simple e intuitiva como la que hemos desarrollado hará que usuarios sin conocimientos avanzados sobre técnicas de minería de datos puedan usar la aplicación sin dificultades.

Todo el desarrollo ha sido un continuo esfuerzo y a la vez reto para nosotros ya que, para empezar, este proyecto es nuevo, es decir, empezamos de cero. Tuvimos que aprender cómo funcionan los algoritmos por dentro para saber cómo preprocesar los datos de tal manera que el algoritmo entienda dichos datos y nos devuelva la salida que nosotros deseábamos. Así mismo tuvimos que recopilar información acerca de los demás parámetros que se usan en la invocación de los algoritmos, ya sean de recomendación, de clustering o de clasificación. El principio del proyecto fue bastante duro para todo el grupo ya que no teníamos información alguna y tuvimos que buscar en todos los medios disponibles.

Con solo echar un vistazo a las listas de emails de Hadoop o Mahout se ve la cantidad de fallos y problemas que las personas que utilizan estos frameworks se encuentran, pues se trata de un software bastante joven. Para nosotros tampoco ha sido fácil conseguir que los algoritmos incluidos en nuestra aplicación funcionen correctamente, pasando días buscando soluciones a los problemas, y días enteros esperando que alguno de los expertos desarrolladores de Hadoop ofreciera su ayuda y conclusiones.

En cuanto a los objetivos que nos planteamos al inicio del proyecto podemos concluir que se han cumplido todos, tanto en la parte de los sistemas de recomendación como en la parte de clustering y clasificación. Ofrecemos la posibilidad de ejecutar estos algoritmos tanto de manera local como mediante MapReduce (Hadoop).

Como trabajo futuro, nos gustaría trabajar en:

- Incorporación de los algoritmos restantes implementados en Mahout, en continuo desarrollo, como podrían ser clustering Dirichelet o la clasificación Random Forest.
- Añadir la ejecución no distribuida a los clasificadores para poder ejecutar los algoritmos sobre un solo fichero.
- Mostrar el resultado del clustering gráficamente, mostrando al usuario los puntos y los clústeres a los que pertenecen.
- Permitir la ejecución totalmente distribuida, es decir, con distintos nodos reales. Para ello (y gracias a Hadoop) bastaría con modificar los ficheros de configuración de

Hadoop añadiendo los nodos, y configurar los demonios para que ejecuten los Jobs correctamente en los distintos nodos.

8. Bibliografía

1. http://www.sas.com/big-data/
2. http://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/
3. http://www.ibm.com/big-data/us/en/
4. Oracle White Paper: Big Data for the Enterprise (June 2013, Oracle)
5. Paper: Data Management Controlling Data Volume, Velocity and Variety. (February 2001, Doug Lancy)
6. www.bigdatauniversity.com
7. http://es.wikipedia.org/wiki/Hadoop
8. http://www.norbertogallego.com/un-elefante-ocupa-poco-espacio-dos-elefantes.../2012/04/13/
9. http://momentotic.wordpress.com/2013/05/16/que-es-hadoop/
10. http://www.ticout.com/blog/2013/04/02/introduccion-a-hadoop-y-su-ecosistema/
11. HDFS Java API: http://hadoop.apache.org/core/docs/current/api/
12. HDFS source code: http://hadoop.apache.org/hdfs/version_control.html
13. http://hadoop.apache.org/docs/stable/hdfs_design.html
14. http://hive.apache.org/
15. https://cwiki.apache.org/confluence/display/Hive/Home
16. https://cwiki.apache.org/confluence/display/Hive/LanguageManual+VariableSubstitution
17. http://es.wikipedia.org/wiki/Hadoop
18. http://es.wikipedia.org/wiki/MapReduce
19. http://blogs.solidq.com/bigdata-hadoop/post.aspx?id=13&title=
20. http://unpocodejava.wordpress.com/2013/05/22/explicando-mapreduce/
21. http://www.ibm.com/developerworks/ssa/data/library/bigdata-apachepig/
22. http://chimera.labs.oreilly.com/books/1234000001811/ch12.html#pig_vs_hive
23. Paper: “Large-scale Parallel Collavorative Filtring for Netflix Prize” (Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan)
24. Paper: “Collaborative Filtering for Implicit Feedback Datasets” (Yifan Hu, Yehuda Koren, Chris Volinsky)
25. Paper: “Matrix Factorization Techniques for Recommender Systems”(Yehuda Koren, Robert Bell and Chris Volinsky)
26. Libro: Hadoop: The Definitive Guide 3rd Edition, O’REILLY
27. Libro: Mahout in Action, Manning
28. Libro: Apache Mahout Cookbook, Piero Giacomelli
29. Paper: Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model (Yehuda Koren)
30. http://ecomcanada.wordpress.com/2012/11/14/storing-and-querying-big-data-in-hadoop-hdfs/
31. http://inside-bigdata.com/2014/05/01/data-science-101-introduction-mapreduce/
32. http://www.ticout.com/blog/2013/04/02/introduccion-a-hadoop-y-su-ecosistema/